BAB 2
TINJAUAN PUSTAKA

2.1 Kecerdasan Buatan

Kecerdasan buatan atau *Artificial Intelligence* (AI) merupakan cabang dari ilmu komputer yang berhubungan dengan pengautomatisasian tingkah laku cerdas. Kecerdasan buatan didasarkan pada teori suara (*sound theoretical*) dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan, algoritma yang diperlukan untuk mengaplikasikan pengetahuan tersebut serta bahasa dan teknik pemrograman yang digunakan dalam mengimplementasikannya [5].

Berdasarkan sudut pandang, AI dapat dipandang sebagai berikut :

1. Sudut pandang kecerdasan, AI adalah bagaimana membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan manusia.
2. Sudut pandang bisnis, AI adalah sekelompok alat bantu yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah-masalah bisnis.
3. Sudut pandang pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah dan proses pencarian.
4. Sudut pandang penelitian :

a. Riset tentang AI dimulai pada awal tahun 1960-an, percobaan pertama adalah membuat *program* permainan catur, membuktikan teori dan *general problem solving*.
b. AI adalah nama pada akar dari studi area.

2.2 Teknik-teknik Dasar Pencarian

Pencarian atau pelacakan merupakan salah satu teknik untuk menyelesaikan permasalahan AI. Keberhasilan suatu sistem salah satunya ditentukan oleh kesuksesan dalam pencarian dan pencocokan. Teknik dasar pencarian memberikan suatu kunci bagi banyak sejarah penyelesaian yang penting dalam bidang AI. Ada beberapa aplikasi yang menggunakan teknik pencarian ini, yaitu :

1. Papan *game* dan *puzzle* (tic-tac-toe, catur, menara hanoi).
2. Penjadwalan dan masalah *routing (travelling salesman problem)*.
3. *Parsing* bahasa dan inteprestasinya (pencarian struktur dan arti).
4. Logika pemrograman (pencarian fakta dan implikasinya).
5. *Computer vision* dan pengenalan pola.
6. Sistem pakar berbasis kaidah *(rule based expert system)*.

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan *(state space)*. Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin. Kondisi suatu pencarian meliputi :

1. Keadaan sekarang atau awal.
2. Keadaan tujuan-solusi yang dijangkau dan perlu diperiksa apakah telah mencapai sasaran.
3. Biaya atau nilai yang diperoleh dari solusi.

Solusi merupakan suatu lintasan dari keadaan awal sampai keadaan tujuan. Secara umum, proses pencarian dapat dilakukan seperti berikut :

1. Memeriksa keadaan sekarang atau awal.
2. Mengeksekusi aksi yang dibolehkan untuk memindahkan ke keadaan berikutnya.
3. Memeriksa jika keadaan baru merupakan solusinya. Jika tidak, keadaan baru tersebut menjadi keadaan sekarang dan proses ini diulangi sampai solusi ditemukan atau ruang keadaan habis terpakai [5].

2.3 Algoritma Pencarian

Permasalahan pencarian dapat diselesaikan dengan beberapa metode yaitu [5]:

1. Metode pencarian yang pertama adalah metode sederhana yang hanya berusaha mencari kemungkinan penyelesaian yang disebut juga pencarian buta.
2. Metode yang lebih kompleks yang akan mencari jarak terpendek. Metode ini adalah *British Museum Procedure, Branch* and *Bound, Dynamic Programming, Best First Search, Greedy Search, A* (A Star)* dan *Hill Climbing Search*. Metode-metode ini digunakan pada saat perjalanan untuk mencari kemungkinan menjadi perhitungan.

Metode pencarian sangat penting untuk menyelesaikan permasalahan karena setiap *state* atau keadaan menggambarkan langkah-langkah untuk menyelesaikan permasalahan. Dalam sebuah permainan, metode pencarian akan menentukan apa yang harus dilakukan di mana setiap state menggambarkan kemungkinan posisi pada suatu saat. Metode pencarian adalah bagian dari kesimpulan di mana setiap *state* menggambarkan hipotesis dalam sebuah rangkaian deduktif.

Menurut cara algoritma mengembangkan *node* dalam proses pencarian, gambar bagan metode penelusuran dibagi menjadi dua golongan, yakni pencarian buta (*blind search*) dan pencarian terbimbing (*heuristic search*).

Beberapa contoh algoritma pencarian yang menggunakan metode *heuristic search* adalah : *Best First Search, Greedy Search, A* (A Star) Search*, dan *Hill Climbing Search.*

2.3.1 Pencarian Buta (*Blind Search*)

Pencarian buta (*Blind Search*) adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state*. Informasi yang ada hanyalah definisi *goal state* itu sendiri sehingga algoritma dapat mengenali *goal state*. Apabila tidak ada informasi maka pencarian buta dalam kerjanya akan memeriksa *node-node* secara tidak terarah dan kurang

efisien untuk kebanyakan kasus karena banyaknya *node* yang dikembangkan. Beberapa contoh algoritma yang termasuk *blind search* adalah *Breadth First Search, Uniform Cost Search, Depth First Search, Depth Limited Search, Interative Deepening Search* dan *Bidirectional Search* [5].

2.3.2 Pencarian Terbimbing (*Heuristic Search*)

Pencarian terbimbing (*Heuristic Search*) mempunyai informasi tentang biaya untuk mencapai *goal state* dari *current state*. Pencarian terbimbing dapat melakukan pertimbangan untuk mengembangkan atau memeriksa *node-node* yang mengarah ke *goal state*. Pencarian terbimbing untuk menghitung *cost* ke *goal state* digunakan fungsi *heuristic*. Fungsi *heuristic* berbeda daripada algoritma di mana *heuristic* lebih merupakan perkiraan untuk membantu algoritma dan tidak harus *valid* setiap waktu. Beberapa contoh algoritma pencarian yang menggunakan metode *heuristic search* adalah *Best First Search, Greedy Search, A* (A Star)* dan *Hill Climbing Search* [5].

2.4 Algoritma A* (*A* Star)

Algoritma A* merupakan perbaikan dari metode BFS dengan memodifikasi fungsi *heuristicnya*. A* (A *Star*) akan meminimumkan total biaya lintasan. Pada kondisi yang tepat, A* akan memberikan solusi yang terbaik dalam waktu yang optimal [5].

Pada pencarian rute kasus sederhana, di mana tidak terdapat halangan pada peta, A* bekerja secepat dan seefisien BFS. Pada kasus peta dengan halangan, A* dapat menemukan solusi rute tanpa "terjebak" oleh halangan yang ada.

Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS, hanya saja dengan dua faktor tambahan.

1. Setiap sisi mempunyai "*cost*" yang berbeda-beda, seberapa besar *cost* untuk pergi dari satu simpul ke simpul yang lain.
2. *Cost* dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan kita mencari ke arah yang salah.

Cost untuk setiap simpul tidak harus berupa jarak. *Cost* bisa saja berupa waktu bila kita ingin mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila kita berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit.

Algoritma A* bekerja dengan prinsip yang hampir sama dengan BFS, kecuali dengan dua perbedaan, yaitu :

1. Simpul-simpul di *list* "terbuka" diurutkan oleh *cost* keseluruhan dari simpul awal ke simpul tujuan, dari *cost* terkecil sampai *cost* terbesar. Dengan kata lain, menggunakan *priority queue* (antrian prioritas). *Cost* keseluruhan dihitung dari *cost* dari simpul awal ke simpul sekarang (*current node*) ditambah *cost* perkiraan menuju simpul tujuan.
2. Simpul di *list* "tertutup" bisa dimasukkan ke *list* "terbuka" bila jalan terpendek (*cost* lebih kecil) menuju simpul tersebut ditemukan.

Karena *list* "terbuka" diurutkan berdasarkan perkiraan *cost* keseluruhan, algoritma mengecek simpul-simpul yang mempunyai perkiraan *cost* yang paling kecil terlebih dahulu, jadi algoritmanya mencari simpul-simpul yang kemungkinan mengarah ke simpul tujuan. Karena itu, lebih baik perkiraan *cost*-nya, lebih cepat pencariannya. *Cost* dan perkiraannya ditentukan oleh kita sendiri. Bila *cost*-nya adalah jarak, akan menjadi mudah.

Cost antara simpul adalah jaraknya, dan perkiraan *cost* dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Atau agar lebih mudahnya bisa ditunjukkan seperti berikut ini.

$$f(n) = g(n) + h(n) \qquad (2.1)$$

Dengan:

f(n) = fungsi evaluasi

g(n) = biaya (*cost*) yang sudah dikeluarkan dari keadaan sampai keadaan n

h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

Node dengan nilai terendah merupakan solusi terbaik untuk diperiksa pertama kali pada g(n) + h(n). Dengan fungsi *heuristic* yang memenuhi kondisi tersebut maka pencarian dengan algoritma A* dapat optimal. Keoptimalan dari

A* cukup langsung dinilai optimal jika h(n) adalah *admissible heuristic* yaitu nilai h(n) tidak akan memberikan penilaian lebih pada *cost* untuk mencapai tujuan. Salah satu contoh dari *admissible heuristic* adalah jarak dengan menarik garis lurus karena jarak terdekat dari dua titik adalah dengan menarik garis lurus [6].

Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point, current node, simpul, neighbor node, open set, closed set, came from,* harga (*cost*), *walkability, target point*.

1. Simpul awal adalah sebuah terminologi untuk posisi awal sebuah benda.
2. *Current node* adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek.
3. Simpul adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga.
4. *Neighbour node* adalah simpul-simpul yang bertetangga dengan *current node*.
5. *Open List* adalah tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan.
6. *Closed List* adalah tempat menyimpan data simpul sebelum *current node* yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.
7. *Parent* adalah tempat menyimpan data ketetanggaan dari suatu simpul, misalnya y *parent* x artinya *neighbour node* y dari *current node* x.
8. Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke *current node*, dan H, jumlah nilai perkiraan dari sebuah simpul ke target *point*.
9. *Target point* yaitu simpul yang dituju.
10. *Walkability* adalah sebuah atribut yang menyatakan apakah sebuah simpul dapat atau tidak dapat dilalui oleh *current node*.

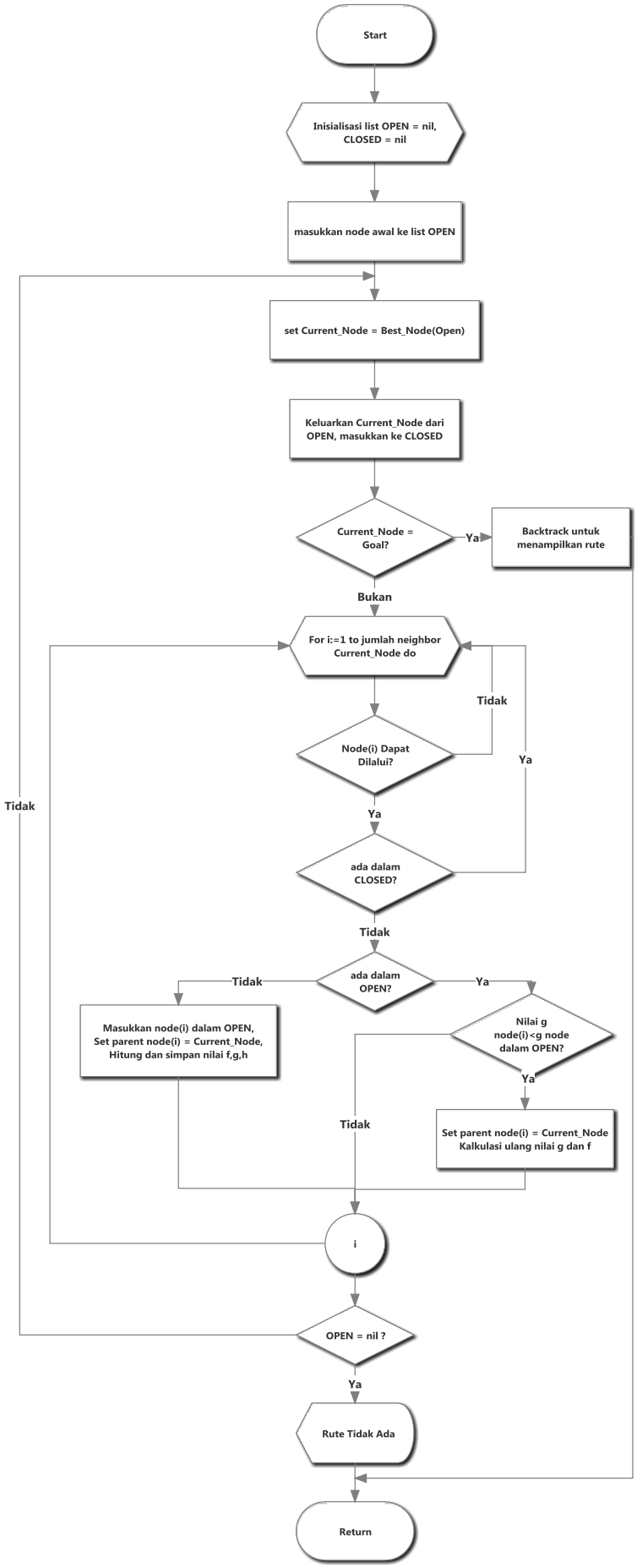
Algoritma A* secara ringkas langkah demi langkahnya adalah sebagai berikut:

1. Tambahkan simpul awal ke dalam *open list*.
2. Ulangi langkah berikut sampai pencarian berakhir:

a. Carilah simpul n dengan biaya F(n) paling rendah, dalam *open* list. Simpul dengan biaya F terendah kemudian disebut *current node*.
b. Keluarkan *current node* dari *open list* dan masukkan ke dalam *closed list*.
c. Untuk setiap 8 simpul (*neighbor node*) dari *current node* lakukan langkah berikut:
    1). Jika sudah terdapat dalam *closed list* atau tidak *walkable*, maka abaikan, jika tidak lanjutkan.
    2). Jika belum ada pada *open list*, tambahkan ke *open list*. Simpan *current node* sebagai *parent* dari *neighbor node* ini. Simpan harga F masing-masing simpul.
    3). Jika sudah ada dalam *open list*, periksa apakah ini jalan dari simpul ini ke *current node* yang lebih baik dengan menggunakan biaya G sebagai ukurannya. Simpul dengan biaya G yang lebih rendah berarti bahwa ini adalah jalan yang lebih baik. Jika demikian, buatlah simpul ini (*neighbor node*) sebagai *parent* dari *current node*, dan menghitung ulang nilai G dan F dari simpul ini.
d. Berhenti ketika:
    1. Menambahkan *target point* ke dalam *closed list*, dalam hal ini jalan telah ditemukan, atau,
    2. Gagal untuk menemukan *target point*, dan *open list* kosong. Dalam kasus ini, tidak ada jalan.
e. Walaupun telah mencapai *target point*, jika masih ada *neighbor node* yang memiliki nilai yang lebih kecil, maka simpul tersebut akan terus dipilih sampai bobotnya jauh lebih besar atau mencapai *target point* dengan bobot yang lebih kecil dibanding dengan simpul sebelumnya yang telah mencapai *target point*.
f. Pada saat pemilihan simpul berikutnya, nilai F(n) akan dievaluasi, dan jika terdapat nilai F(n) yang sama maka akan dipilih berdasarkan nilai G(n) terbesar.

3. Simpan jalan. Bekerja mundur dari *target point*, pergi dari masing-masing simpul ke simpul *parent* sampai mencapai *starting point*.

Adapun *flowchart* untuk algoritma A* dapat dilihat pada gambar 2.1



Gambar 2.1 *Flowchart* algoritma A*

2.4.1 Kompleksitas Algoritma A* (A Star)

Kompleksitas waktu dari algoritma A* tergantung dari heuristiknya. Dalam kasus terburuk (*worst case*), jumlah simpul yang diekspansi bisa eksponensial dalam solusinya (jalan terpendek). Akan tetapi, kompleksitasnya bisa berupa *polynomial* bila fungsi *heuristic* h bertemu kondisi berikut:

$$|h(x) - h^{x}(x)| = O(\log h^{*}(x)) \qquad (2.2)$$

Dimana h* adalah heuristik optimal, atau *cost* pasti untuk menuju tujuan dari x. Dengan kata lain, kesalahan (*error*) dari h tidak boleh tumbuh lebih cepat dari algoritma "*perfect heuristic*" h* yang mengembalikan jarak sebenarnya dari x menuju tujuan. [1].

2.4.2 Efisien Waktu Algoritma A*

Dengan digunakannya fungsi *heuristic* H(n). algoritma A* dapat memfokuskan pencarian pada *node* - *node* yang berada pada arah yang mendekati tujuan. Kemudian pencarian diterminasikan pada waktu *node* tujuan diperiksa. Hal ini dapat meminimalisasikan jumlah *node* yang harus diperiksa dan arena waktu yang diperlukan untuk mendapatkan jalur berbanding lurus dengan jumlah *node* yang diperiksa, maka waktu pencarian dapat diminimalisasikan.

Walaupun jumlah *node* yang diperiksa dapat diminimalisasikan, algoritma A* mempunyai kasus terburuk. Pada kasus ini, sebagian besar ataupun keseluruhan *node* pada jalan diperiksa, sehingga algoritma A* bekerja seperti algoritma dijkstra atau BFS (*Best-First-Search*). Ada dua hal yang dapat menyebabkan keadaan terburuk ini, yaitu keadaan sepadan dan jika jalur yang dicari tidak ditemukan. [1]

2.5 Fungsi Heuristik

BFS dan A* sebagai algoritma pencarian yang menggunakan fungsi heuristik untuk "menuntun" pencarian rute, khususnya dalam hal pengembangan dan pemeriksaan *node* - *node* pada peta. Dalam aplikasi ini, fungsi heuristik yang dipakai untuk pencarian rute mengisi nilai/notasi h pada algoritma BFS dan A*.

Ada beberapa fungsi heuristik umum yang bisa dipakai untuk algoritma BFS dan A* ini. Salah satunya adalah yang dikenal dengan istilah "*Manhattan Distance*". Fungsi heuristik ini digunakan untuk kasus di mana pergerakan pada peta hanya lurus (*horizontal* atau *vertikal*), tidak diperbolehkan pergerakan *diagonal*. [1]

Perhitungan nilai heuristik untuk *node* ke-n menggunakan *Manhattan Distance* adalah sebagai berikut :

$$\mathrm{h(n)} = \left(\mathrm{abs(n.x} - goal.\mathrm{x}) + \mathrm{abs(n.y} - goal.\mathrm{y})\right) \qquad (2.3)$$

Dimana h(n) adalah nilai heuristik untuk *node* n, dan *goal* adalah *node* tujuan.

Jika pergerakan *diagonal* pada peta diperbolehkan, maka digunakan fungsi heuristik selain *Manhattan Distance*. Untuk mendekati kenyataan, *cost* untuk perpindahan *node* secara *diagonal* dan *orthogonal* dibedakan. *Cost diagonal* adalah 1,4 kali cost perpindahan secara *orthogonal*.

Maka fungsi heuristic yang digunakan adalah sebagai berikut:

$$\mathrm{h}_{diagonal(\mathrm{n})} = \min\left(\mathrm{abs(n.x} - goal.\mathrm{x}) + \mathrm{abs(n.y} - goal.\mathrm{y})\right) \qquad (2.4)$$

$$\mathrm{h}_{orthogonal(\mathrm{n})} = (\mathrm{abs(n.x} - goal.\mathrm{x}) + \mathrm{abs(n.y} - goal.\mathrm{y})) \qquad (2.5)$$

$$\mathrm{h(n)} = \mathrm{h}_{diagonal(\mathrm{n})} + (\mathrm{h}_{orthogonal(\mathrm{n})} - (2 * \mathrm{h}_{diagonal(\mathrm{n})})) \qquad (2.6)$$

Dimana h_*diagonal*(n) adalah banyaknya langkah *diagonal* yang bisa diambil untuk mencapai *goal* dari *node* n. h_*orthogonal* adalah banyaknya langkah lurus yang bisa diambil untuk mencapai *goal* dari *node* n.

Nilai heuristik kemudian diperoleh dari h_*diagonal*(n) ditambah dengan selisih h_*orthogonal*(n) dengan dua kali h_*diagonal*(n). Dengan kata lain, jumlah langkah *diagonal* kali *cost* diagonal ditambah jumlah langkah lurus yang masih bisa diambil dikali *cost* pergerakan lurus [1].

2.6 Simpified Memory-Bounded A*

Simplified memory-bounded A* merupakan pengembangan dari algoritma A* yang mengatasi masalah *storage problem* dengan meletakkan *limit* tetap pada ukuran *open list*. Ketika sebuah *node* baru diproses, jika memiliki nilai yang lebih

besar dari setiap *node* yang ada di *list*, maka akan dibuang. Jika tidak, maka akan ditambahkan, dan *node* yang sudah dalam *list* dengan nilai terbesar dihapus.

Pendekatan ini bisa jauh lebih efisien dibandingkan dengan pendekatan IDA*(*Iterative Deepening* A*), walaupun ini masih bisa menyebabkan mengunjungi ulang node yang sama berulang kali selama proses pencarian. Hal ini sangat tergantung dengan nilai heuristik yang digunakan. Nilai heuristik yang tidak tepat bisa menyebabkan *node* yang penting tersingkirkan oleh *node* yang tidak berguna dari *open list*.

SMA* adalah contoh dari sebuah mekanisme pencarian "*lossy*". Dalam rangka untuk mengurangi konsumsi memori, hal ini membuang informasi, dengan asumsi bahwa informasi yang dibuang itu tidak penting. Bagaimanapun, tidak ada jaminan bahwa hal itu tidak penting. Dalam semua kasus dengan SMA*, jalur yang ditemukan tidak memiliki jaminan menjadi jalur yang optimal. Pada awal pencarian, *node* yang tidak menjanjikan bisa saja dibuang.

Menetapkan *limit* yang besar pada ukuran *open list* dapat membantu meringankan masalah ini, namun fungsi untuk mengurangi penggunaan memori menjadi terbuang. Pada kasus ekstrem yang lain, dengan memberi *limit* 1 simpul pada *open list*, ini bisa mempercepat sekaligus mengurangi penggunaan memori dalam pencarian, namun jalur yang ditemukan bisa tidak optimal [3].

SMA* memiliki beberapa sifat sebagai berikut [7]:

1. F-*cost* digunakan untuk mendapatkan node dengan nilai terkecil diantara semua *node* yang berada di *open list.*
2. SMA* selalu memperluas *node* yang memiliki nilai terbaik pada keturunannya yang belum diperiksa.
3. SMA* dijamin mendapatkan solusi yang optimal, jika disediakan *max* setidaknya sama besar dengan jumlah *node* pada jalur solusi optimal.
4. SMA* berperilaku identik dengan A* bila *max* lebih besar dari jumlah *node* yang dihasilkan oleh A*.

Adapun deskripsi atau *pseucode* dari algoritma SMA* bisa dilihat pada gambar 2.2

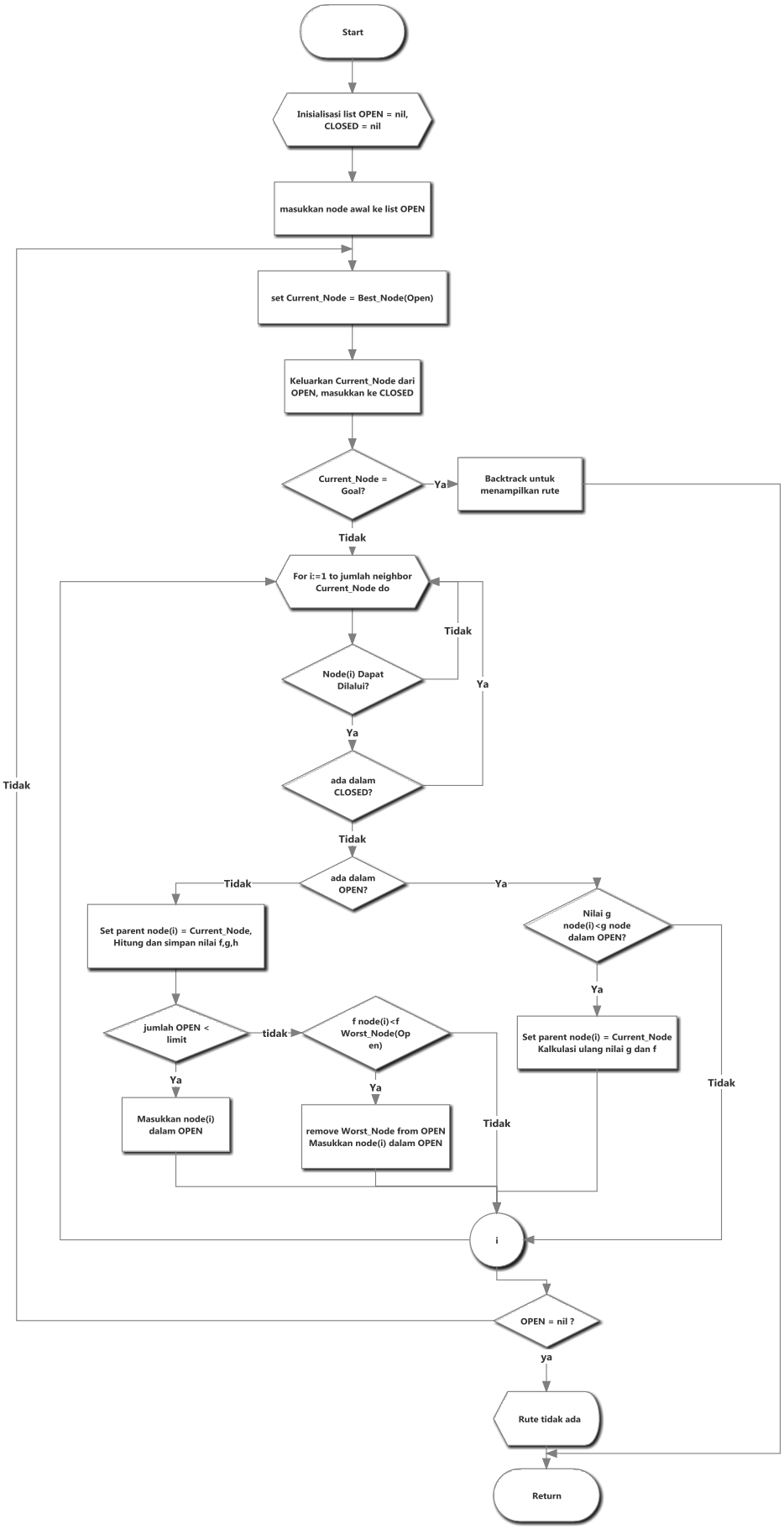
Algorithm SMA*(start):

```
put start on OPEN; USED – 1;
loop
        if empty(OPEN) return with failure;
        best ← deepest least f-cost leaf in OPEN;
        if goal(best) then return with success;
        succ ← next successor (best);
        f(succ) ← max(f(best),g(succ) + h(succ));
        if completed(best), BACKUP(best);
        if S(best) all in memory, remove best from OPEN
        USED ← USED +1;
        if USED > MAX then
                delete shallowest, highest f-cost node in OPEN;
                remove it from its parent's successor list;
                insert its parent on OPEN if necessary;
                USED ← USED -1;
        insert succ on OPEN.

Procedure BACKUP(n)
if n is completed and has parent then
        f(n)=least f cost of all successors;
        if f(n) changed, BACKUP(parent(n));
```

Gambar 2.2 *Pseudocode* Algoritma SMA* [7]

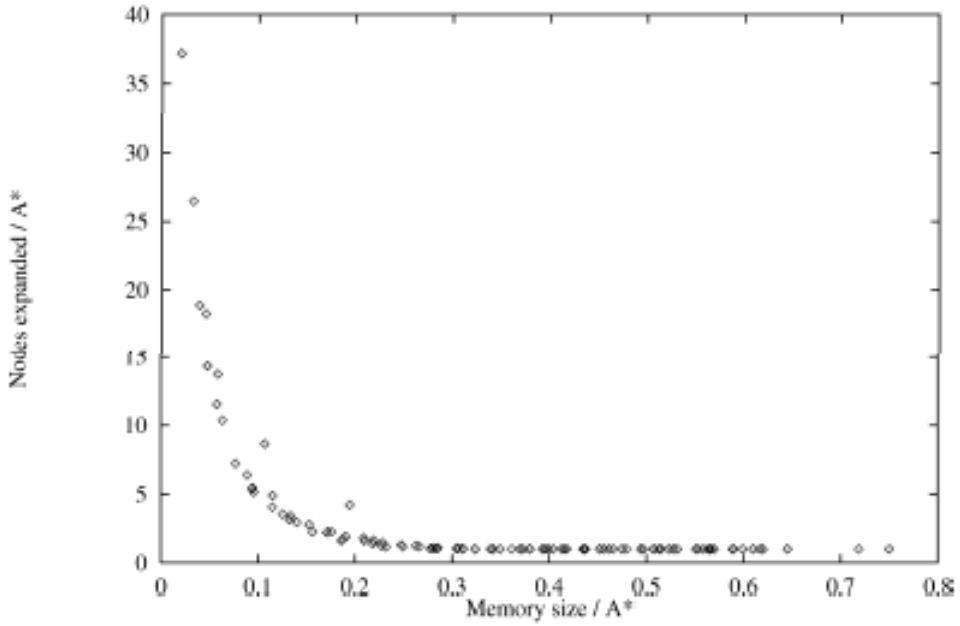
SMA* dimulai dengan memasukkan *node* awal ke dalam *open list. Variabel global max* ditentukan untuk maksimum *node* yang bisa dimasukkan ke dalam *open list*, dan variabel *used* digunakan untuk melihat berapa banyak *node* yang sedang berada di dalam memori. Di dalam setiap *node* disimpan nilai g, h, dan f-*costs*, dan juga disimpan informasi mengenai *parent* dari *node* tersebut. [7] Untuk *flowchart* dari algoritma SMA* bisa dilihat pada gambar 2.3



Gambar 2.3 *Flowchart* algoritma SMA*

2.6.2 Efisiensi Algoritma SMA*

Salah satu perbedaan utama antara A* dan SMA* adalah penggunaan fungsi *backup*. Seperti ditunjukkan pada gambar 2.2 SMA* menggunakan fungsi *backup* ketika memperluas *node*. Fungsi ini secara rekursif meng-*update* nilai f(n) dari parent. *Running time* kasus terburuk untuk fungsi ini adalah O(n), n adalah kedalaman simpul dari *node* awal. Ini jauh lebih mahal daripada biaya ekspansi A*. SMA. gambar 2.4 menunjukkan jumlah re-ekspansi yang dilakukan oleh SMA* dibandingkan dengan SMA*. Garis *vertikal* pada gambar menunjukkan jumlah *node* yang diperluas oleh SMA* dibandingkan dengan jumlah *node* yang diperluas oleh A*. Jika garis *vertikal* menunjukkan 25 maka itu berarti SMA* diperluas 25 kali lebih besar dibandingkan A*. Garis horisontal pada gambar menunjukkan persentase memori yang digunakan oleh SMA* untuk memecahkan masalah yang sama dibandingkan dengan A*. Sebagai contoh, jika titik garis vertikal 25 dan titik garis *horizontal* 0.05 maka berarti SMA* bisa memecahkan masalah yang sama dengan menggunakan 5% dari memori yang digunakan oleh A* tapi harus melakukan ekspansi 25 kali lebih banyak dari A*. [7]



Gambar 2.4 SMA* vs A* [7]

2.7 Linked List

Dikembangkan tahun 1955-1956 oleh Allen Newell, Cliff Shaw dan Herbert Simon di RAND *Corporation* sebagai struktur data utama untuk bahasa *Information Processing Language* (IPL). IPL dibuat untuk mengembangkan program *artificial intelligence*, seperti pembuatan *Chess Solver*. Victor Yngve di Massachusetts Institute of Technology (MIT) juga menggunakan *linked list* pada *natural language processing* dan *machine transitions* pada bahasa pemrograman COMMIT.

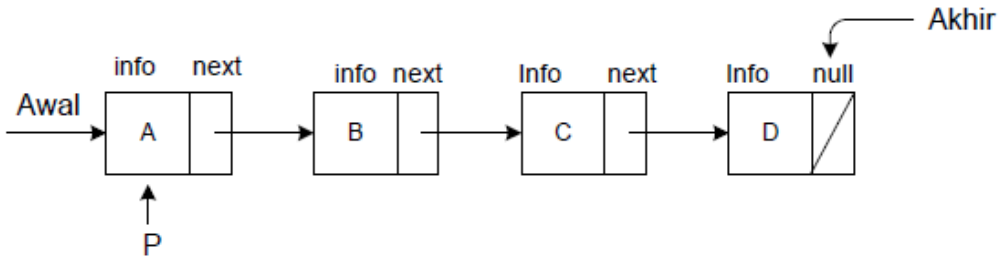
Linked list adalah salah satu bentuk struktur data, berisi kumpulan data (*node*) yang tersusun secara sekuensial, saling sambung menyambung, dinamis dan tidak terbatas. Pada tabel 2.1 dapat disimpulkan *linked list* lebih optimal daripada *array*. *Linked list* sering disebut juga senarai berantai dan saling terhubung dengan bantuan *variabel pointer*. Masing-masing data dalam *linked list* disebut dengan *node* (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa *struct* yang terdiri dari beberapa *field*.

Tabel 2.1 Perbandingan *array* dan *linked list*

Array	*Linked List*
Statis	Dinamis
Penambahan/penghapusan data terbatas	Penambahan/penghapusan data tidak terbatas
Random access	*Random access*
Penghapusan *array* tidak mungkin	Penghapusan *linked list* mudah

Linked list terdiri dari tiga jenis yaitu *single linked list*, *double linked list* dan *circular linked list*. Salah satu yang akan dijelaskan yaitu tentang *single linked list*.

Single linked list adalah *linked list* dengan simpul berisi satu *link/pointer* yang mengacu ke simpul berikutnya, ilustrasinya dijelaskan pada gambar 2.5.



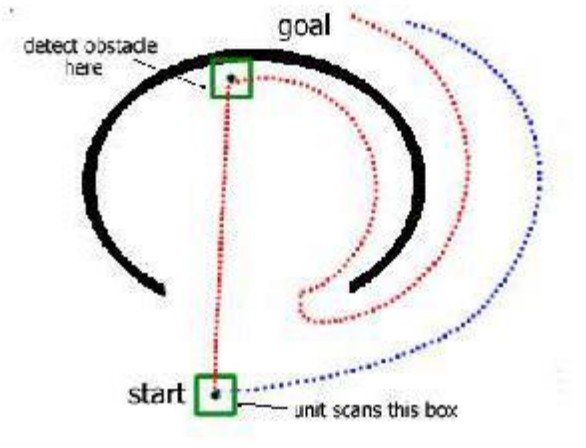
Gambar 2.5 Ilustrasi *single linked list* [8]

Setiap *node* pada *linked list* mempunyai *field* yang berisi *pointer* ke *node* berikutnya, dan juga memiliki *field* yang berisi data. Pada akhir *linked list*, *node* terakhir akan menunjuk ke *NULL* yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi *linked list*.

Penambahan *node* baru memiliki tiga cara penyisipan yaitu sisip didepan, sisip ditengah dan sisip diakhir. Pada pembahasan ini akan lebih dijelaskan tentang sisip didepan, penambahan data didepan akan dikaitan di *node* paling awal, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan dengan cara "Awal" ditunjukkan ke *node* baru tersebut [8].

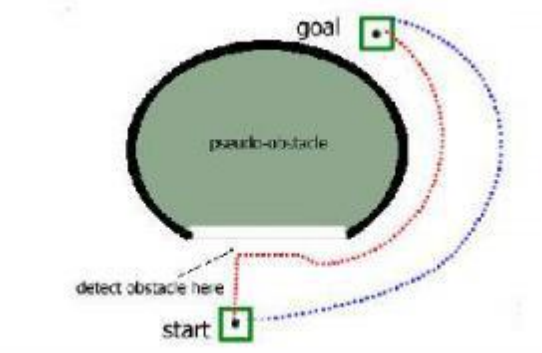
2.8 Pathfinding

Pathfinding (pencarian jalan/rute) adalah salah satu bidang penerapan yang sering ditangani oleh kecerdasan buatan khususnya dengan menggunakan algoritma pencarian. Penerapan yang dapat dilakukan dengan *pathfinding* antara lain adalah pencarian rute dalam suatu *game* dan pencarian jalan/rute pada suatu peta. Algoritma pencarian yang dipakai harus dapat mengenali jalan dan elemen peta yang tidak dapat dilewati. Sebuah algoritma pathfinding yang baik dapat bermanfaat untuk mendeteksi halangan/rintangan yang ada pada medan dan menemukan jalan menghindarinya, sehingga jalan yang ditempuh lebih pendek daripada yang seharusnya bila tidak menggunakan algoritma *pathfinding*. Lihat ilustrasi pada gambar 2.6.



Gambar 2.6 Penentuan rute tanpa *pathfinding*[4]

Pada gambar 2.6, dari start menuju gol, tanpa algoritma *pathfinding*, unit hanya akan memeriksa lingkungan sekitarnya saja (dilambangkan dengan daerah di dalam kotak hijau). Unit tersebut akan maju terus ke atas untuk mencapai tujuan, baru setelah mendekati adanya halangan, lalu berjalan memutar untuk menghindarinya. Sebaliknya, penentuan *rule* dengan algoritma *pathfinding* pada gambar 2.7 akan memprediksi ke depan mencari jalan yang lebih pendek menghindari halangan (dilambangkan garis biru) untuk mencapai tujuan, tanpa pernah mengirim unit ke dalam 'perangkap' halangan berbentuk U. Karena itu peran algoritma *pathfinding* sangat berguna untuk memecahkan berbagai permasalahan dalam penentuan rute.



Gambar 2.7 Penentuan rute dengan *pathfinding*[4]

2.9 Multimedia

Multimedia adalah media yang menggabungkan dua unsur atau lebih media yang saling berhubungan diantara satu dengan yang lainnya seperti teks, grafik, gambar, *audio, video* dan animasi. Terdapat dua kategori multimedia yaitu [9]:

1. Multimedia Linier merupakan multimedia yang tidak dilengkapi dengan alat pengontrol yang dapat dioperasikan oleh pemain. Contohnya TV dan Film.
2. Multimedia Interaktif merupakan multimedia yang dilengkapi alat pengontrol yang dapat dioperasikan oleh pemain sehingga dapat memilih apa yang dikehendaki untuk proses selanjutnya. Contohnya multimedia pembelajaran interaktif, aplikasi *game* dan lain-lain.

Menurut beberapa ahli multimedia memiliki beberapa definisi yaitu [7]:

1. Turban dan kawan-kawan

   Multimedia adalah kombinasi dari paling sedikit dua media *input* dan *output*. Media ini dapat berupa *audio*, animasi, *video*, teks, grafik dan gambar.

2. Robin dan Linda

   Multimedia adalah alat yang dapat menciptakan presentasi yang dinamis dan interaktif yang mengkombinasikan teks, grafik, animasi, *audio* dan *video*.

3. Hofstetter

   Multimedia adalah pemanfaatan komputer untuk membuat dan menggabungkan teks, grafik, *audio, video* dengan menggunakan *tool* yang memungkinkan pemakai berinteraksi, berkreasi dan berkomunikasi.

   Dalam definisi ini terdapat empat komponen multimedia yaitu :

a. Harus ada peralatan elektronik yang dapat mengkoordinasikan apa yang dapat dilihat dan didengar dan yang dapat berinteraksi dengan pemain.
b. Harus ada *link* yang dapat menghubungkan pemain dengan informasi.
c. Harus ada alat navigasi yang dapat memandu pemain untuk menjelajahi jaringan informasi yang saling terhubung.
d. Harus dapat menyediakan tempat kepada pemain untuk mengumpulkan, memproses dan mengkomunikasikan informasi dan ide dari pemain itu sendiri.

Penggunaan multimedia dapat digunakan di beberapa bidang seperti:

1. Bidang periklanan yang efektif dan interaktif.
2. Bidang pendidikan dalam penyampaian bahan pengajaran secara interaktif dan dapat mempermudah pembelajaran karena didukung oleh berbagai aspek seperti suara, *video*, animasi, teks dan gambar.
3. Bidang jaringan dan internet yang dapat membantu dalam pembuatan *website* yang menarik, informatif dan interaktif.

2.9.1 Jenis Multimedia

Multimedia terdiri dari beberapa jenis yaitu [9]:

1. Teks

Teks merupakan dasar dari pengolahan kata dan informasi berbasis multimedia. Teks mudah disimpan dan membentuk kata, surat atau narasi dalam multimedia. Kebutuhan teks tergantung pada kegunaan aplikasi multimedia. *File* teks memiliki struktur linier sederhana. Dalam multimedia, teks sangat banyak digunakan karena teks sangat efektif untuk menyampaikan ide serta memberikan panduan kepada pemain.

2. Image

Image atau gambar merupakan presentasi multimedia yang lebih menarik perhatian dan dapat mengurangi kebosanan dibandingkan dengan teks. Gambar dapat meringkas dan menyajikan data kompleks dengan cara yang baru dan lebih berguna.

Berikut nilai intensitas warna pada suatu *pixel* yaitu :

a. 1 bit : *binary value image* (0-1).
b. 8 bits : *gray level* (0-255).
c. 16 bits : *high color* (216).
d. 24 bits : 224 *true color*.
e. 32 bits : *true color* (232).

3. Audio

Fenomena fisik yang dihasilkan oleh benda yang berupa sinyal *analog* dengan amplitudo yang berubah secara kontinyu terhadap waktu. Pola isolasi yang terjadi dinamakan sebagai gelombang. Gelombang yang memiliki pola yang berulang pada interval tertentu disebut periode sedangkan frekuensi adalah banyaknya periode dalam 1 detik.

4. Video

Teknologi untuk menangkap, merekam, memproses, mentransmisikan dan menata ulang gambar bergerak. Biasanya menggunakan film *seluloid*, sinyal elektronik atau media digital. Digital *video* adalah jenis *video recording* yang bekerja menggunakan sistem digital dibandingkan dengan *analog* dalam hal representasi *video*-nya. *Video* kamera menggunakan 2 teknik yaitu :

a. *Interlaced* adalah untuk menampilkan gambar dalam *raster scanned display device* seperti CRT dan telivisi *analog*.
b. *Progressive scan* adalah metode untuk menampilkan, menyimpan dan memancarkan gambar dimana setiap baris untuk setiap *frame* digambar secara berurutan.

5. Animasi

Animasi merupakan *illusion of motion* yang dibuat dari *image* statis yang ditampilkan secara berurutan. Animasi mengacu kepada teknik dimana setiap *frame* dalam film dibuat secara terpisah. *Frame* dapat dihasilkan dari komputer, fotografi atau gambar lukisan.

2.10 Game

Game merupakan salah satu media hiburan yang paling popular untuk semua kalangan usia. Sejak pertama kali ditemukan sampai saat sekarang, teknologi *game* telah mengalami kemajuan yang terbilang pesat. Hal ini ditandai dengan berkembangnya jenis *game*, produk, alat dan jenis interaksi *game* dengan penggunaan yang semakin beragam bentuknya.

2.10.1 Pengertian Game

Game merupakan sebuah bentuk seni dimana pemainnya disebut dengan pemain (*player*), diharuskan membuat keputusan-keputusan dengan tujuan mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain (*token*) miliknya untuk mencapai tujuan tertentu. *Video game* adalah bentuk *game* yang interaksi umumnya melibatkan media *video* dan *audio*.

Menurut Andang Ismail terdapat dua pengertian *game* (permainan). Pertama, *game* (permainan) adalah sebuah aktifitas bermain yang murni mencari kesenangan tanpa mencari menang atau kalah. Kedua, permainan diartikan sebagai aktifitas bermain yang dilakukan dalam rangka mencari kesenangan dan kepuasan, namun ditandai pencarian menang – kalah.

Berdasarkan representasinya, *game* dapat dibedakan menjadi 2 jenis yaitu *game* 2 dimensi (2D) dan 3 dimensi (3D). *Game* 2D adalah *game* yang secara matematis hanya melibatkan 2 elemen koordinat kartesius yaitu x dan y,sehingga konsep kamera pada *game* 2D hanya menentukan gambar pada *game* yang dapat dilihat oleh pemain. Sedangkan *game* 3D adalah *game* yang selain melibatkan elemen x dan y juga melibatkan elemen z pada perhitungannya sehingga konsep kamera pada *game* 3D benar-benar menyerupai konsep kamera pada kehidupan nyata [10].

2.10.2 *Game* 3D

Pada saat kemampuan proses pada komputer semakin cepat, para *developer* juga senantiasa menciptakan *game* yang semakin canggih sehingga muncul *engine* dengan grafik 3D (3 dimensi). Maka dari itu, *game* 3D dengan hitungan *polygon* yang sangat besar dan pencahayaan yang sudah canggih, juga tekstur *maping* mulai diproduksi. *Game* 3D merepresentasikan objek dalam bentuk 3 dimensi sehingga objek akan terlihat lebih nyata seperti dalam kehidupan nyata.

Game bertipe 3 dimensi merupakan *game* dengan grafis yang baik dalam penggambaran secara realita, kebanyakan *game-game* ini memiliki perpindahan kamera *(angle)* hingga 360 derajat sehingga dapat melihat secara keseluruhan dunia *game* tersebut [11].

2.10.3 Jenis-jenis *Game*

Berikut ini beberapa jenis *game* berdasarkan cara pembuatannya,cara pemasarannya dan mesin yang menjalankannya. Jenis-jenis *game* tersebut adalah [12]:

1. *Game* PC

Game PC adalah *game* yang dimainkan pada PC (*Personal Computer*) yang memiliki kelebihan yaitu tampilan antarmuka yang baik untuk *input* maupun *output*. *Output* visual berkualitas tinggi karena layar komputer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televisi biasa. Kekurangannya adalah spesifikasi komputer yang sangat bervariasi antar satu komputer dengan komputer yang lainnya menyebabkan beberapa *game* dapat ditampilkan dengan baik pada satu komputer tetapi tidak berjalan dengan baik pada komputer yang lainnya.

2. *Game Console*

Game console adalah *game* yang dijalankan pada suatu mesin spesifik yang biasanya tersedia di rumah seperti Xbox, Nintendo Wii dan lain-lain.

3. *Game Arcade*

Game arcade adalah *game* yang dijalankan pada mesin dengan *input* dan *output* audio visual yang telah terintegrasi dan tersedia ditempat-tempat umum.

4. *Game Online*

Game online adalah *game* yang hanya dapat dumainkan secara *online* melaui LAN atau internet.

2.10.4 Genre Game

Berdasarkan *genre, game* dapat dibagi menjadi beberapa *genre* yaitu [12] :

1. *Action Game*

Action game dikategorikan sebagai *game play* dengan model pertarungan. Berikut beberapa macam *game* yang termasuk dalam *genre action game* yaitu :

a. *Action Adventure Game*

Genre game yang berfokus pada eksplorasi dan biasanya mempunyai unsur *item gathering*, penyelesaiaan *puzzle simple* dan pertarungan. Contoh *game* dari *genre* ini adalah The Legend Of Zelda series dan Metroid series.

b. *Stealth Game*

Termasuk dalam *genre* terbaru,biasanya digolongkan dalam mata-mata yang bias melakukan aksinya secara rahasia. Contoh *game* dari *genre* ini adalah Metal Gear series.

c. *Survival Horror Game*

Genre game yang berusaha membuat pemain menjadi tegang dan takut dengan elemen-elemen horror. Contoh *game* dari *genre* ini adalah Resident Evil series dan Alone in The Dark.

d. *Beat'em Up Game*

Genre game combat dimana satu orang melawan banyak musuh yang telah disediakan.Contoh *game* dari *genre* ini adalah Dynasty Warrior series dan Final Fight.

e. *Fighting Game*

Game pertarungan dua pemain dengan jurus-jurus yang biasa dikeluarkan dengan menekan beberapa tombol pada *keyboard* dengan urutan tertentu. Contoh *game* dari *genre* ini adalah Street Fighter dan Tekken series.

f. *Maze Game*

Genre game yang membutuhkan kecepatan berpikir dan bereaksi serta berunsur ketepatan menavigasi. Contoh *game* dari *genre* ini adalah Pac-Man.

g. *Platfrom Game*

Genre Game dengan *game play* berlari, melompat, mengayun dan sebagainya. Contoh *game* dari *genre* ini adalah Donkey Kong dan Ray Man.

h. *Shooter*

a) *First Person Shooter Game*

Genre game yang mengutamakan *shooting* dan *combat* dari perspektif langsung mata karakter yang bertujuan untuk memberikan pemain perasaan berada ditempat itu dan bisa fokus menembak.

b) *Massively Multiplayer Online First Person Shooter Game*

Genre game yang mengkombinasikan *game play first person shooter* dengan dunia virtual dimana banyak *player* juga ikut bermain melaui internet. Contoh *game* dari *genre* ini adalah Counter Strike *Online*.

c) *Third Person Shooter Game*

Genre game yang sama seperti *first person shooter game* yaitu mengutamakan *shooting* dan *combat* dari perspektif karakter yang bertujuan untuk memberikan pemandangan yang lebih luas dan gerakan yang lebih banyak.

d) *Tactical Shooter Game*

Genre yang mengutamakan perencanaan dan kerja sama tim untuk memenangkan *game*. Contoh *game* dari *genre* ini adalah Tom Clancy's Ghost Recon series.

e) *Light Gun Game*

Genre dengan lebih banyak pada *arcade* dengan peralatan tertentu seperti senjata mainan yang mempunyai sensor khusus terhadap layar. Contoh *game* dari *genre* ini adalah Time Crisis dan Duck Hunt.

f) *Shoot'em Up Game*

Genre dengan ciri khas gambar 2D dan *scrolling playing area*. Contoh *game* dari *genre* ini adalah Star Fox series.

2. *Adventure Game*

Adventure game dikategorikan sebagai *game play* yang mengharuskan pemain memecahkan bermacam-macam teka-teki melaui interaksi dengan orang lingkungan dalam *game* tersebut.

a. *Text Adventure*

Pemain akan menggunakan *keyboard* untuk mengetikkan berupa perintah dan komputer akan menganalisa perintah tersebut lalu menjalankan karakter sesuai perintah tersebut.

b. *Graphical Adventure Game*

Genre yang merupakan perkembangan dari *text adventure*. Pemain dapat menggunakan *mouse* untuk menggerakkan karakter.

c. *Visual Novel Game*

Genre yang memberikan keleluasaan untuk memilih jalan ceritanya sendiri.

d. *Interactive Movie Game*

Genre game dengan rangkaian *live action* dari karakter yang dimainkan pemain. Contoh *game* dari *genre* ini adalah Space Ace.

e. *Dialog Game*

Pada *genre* ini,pemain akan mengalami kemajuan tergantung pada apa yang mereka katakan. Contoh *game* dari *genre* ini adalah Law And Order: The Vangeful Heart.

3. *Role Playing Game*

Role playing game adalah *game* yang memiliki *game play* dimana karakter milik *player* akan berpetualang dengan *skill combat* dalam cerita *game*.

a. *Action Role Playing Game*

Genre game yang memasukkan unsur *action game* dan *action adventure game*. Contoh *game* dari *genre* ini adalah Diablo 1 & 2.

b. *Massively Multyplayer Online Role Playing Game*

Konsep dari *genre* ini terkombinasi dengan *genre-genre* lainnya yang berupa fantasi. Contoh *game* dari *genre* ini adalah Rising Force *Online*.

c. *Tactical Role Playing Game*

Dalam *genre* ini,pemain akan diberikan giliran masing-masing untuk menentukan langkah-langkah yang akan dilakukan oleh karakter. Contoh *game* dari *genre* ini adalah Final Fantasy Tactics.

4. *Simulation Game*

Genre ini bertujuan untuk memberikan pengalaman simulasi kepada pemain.

a. *Construction and Management Simulation Game*

Genre ini merupakan bagian dari *economic simulation game*. Contoh *game* dari *genre* ini adalah Sims City series.

b. *Economic Simulation Game*

Genre ini berupa simulasi keadaan ekonomi dimana pemain mengontrol keadaan ekonomi dari *game* tersebut. Contoh dari *genre* ini adalah Monopoly Tycoon.

c. *God Game*

Dalam *genre* ini tidak ada tujuan akhir yang membuat pemain memenangakan *game*. Contoh *game* dari *genre* ini adalah The Sims series.

d. *Government Simulation Game*

Genre game yang memasukkan unsur kepolisian,pemerintahan atau politik sebuah negara.

5. *Strategy Game*

Genre strategy game berfokus pada *game play* dimana dibutuhkan pemikiran yang tepat agar dapat meraih kemenangan.

a. *Real Time* Strategi

Dalam *real time* strategi, *action* dilakukan dalam waktu yang bersamaan oleh masing-masing pihak dimana *action* dimainkan per *ronde* atau bergiliran. Contoh *game* dari *genre* ini adalah Warcraft series.

b. *Tactical Game*

Dalam *genre* ini pemain harus menggunakan bermacam-macam taktik dan strategi untuk mencapai kemenangan. Contoh *game* dari *genre* ini adalah Dark Omen.

c. *4X Game*

Genre ini berarti penjelajahan, menjajah dan memusnahkan. Contoh *game* dari *genre* ini adalah Galactic Civizations.

d. *Artillery Game*

Genre game ini biasanya mengikutkan *combat* dengan tank atau tentara militer. Contoh *game* dari *genre* ini adalah Tanarus.

6. *Vehicle Simulation Game*

Genre ini merupakan simulasi yang memberikan pemain sebuah pengalaman realistik dalam mengendarai kendaraan tertentu.

a. *Flight Game*

Dalam *genre* ini,pemain tidak hanya bersimulasi mengontrol pesawat terbang tetapi juga bisa *combat* di udara. Contoh *game* dari *genre* ini adalah Falcon 4.0.

b. *Racing Game*

Genre yang menempatkan pemain sebagai *driver* dengan kendaraan seperti mobil. Contoh *game* dari *genre* ini adalah Need For Speed series.

c. *Space Game*

Genre ini bersifat pertarungan di angkasa luar. Contoh *game* dari genre ini adalah Star Wars dan Homeworld.

d. *Train Game*

Genre ini mensimulasikan yang berhubungan dengan trasnportasi kereta. Contoh dari *genre* ini adalah Rail Simulator.

2.10.5 Unsur Game

Dalam sebuah *game* terdapat unsur-unsur yang melengkapinya. Berikut beberapa unsur dalam suatu *game* yaitu [12]:

1. Warna

   Warna mempunyai kemampuan untuk membuat orang tanggap terhadap semua yang dilihat karena tidak ada seseuat hal bermakna tanpa warna.Warna terlihat sebelum penampakan gambar. Mata manusia tertarik oleh warna pada suatu *level* karena warna dari objek diterima sebelum detail-detail dipisahkan oleh bentuk-bentuk dan garisnya. Warna merah memiliki panjang gelombang yang terpanjang,biru memiliki panjang gelombang yang pendek sedangkan hijau memiliki panjang gelombang menengah. Pada anak-anak cenderung tertarik pada warna-warna yang cerah dan mencolok. Warna-warna yang cerah terutama warna *primer* (merah, kuning, biru) dan warna *sekunder* (orange, ungu, hijau).



Gambar 2.8 Warna *Primer* dan *Sekunder*

2. Komposisi

Komposisi adalah pengaturan segala elemen di dalam sebuah karya desain yang sedemikian rupa dengan tujuan tertentu. Komposisi yang baik adalah komposisi yang mampu memenuhi kebutuhan dan tujuan desain, mudah dipahami dan membentuk kesatuan yang serasi dan harmonis. Kemudian *layout* yaitu perencanaan, penempatan semua unsur mulai dari tulisan, gambar, ilustrasi, teks, nama dan sebagainya dengan pengukuran secara seksama. Komposisi yang sesuai dengan anak-anak adalah komposisi yang sederhana dan tidak menggunakan petunjuk terlalu rumit. Kesederhanaan diwujudkan dengan penggunaan visual 2D dan penerapan warna-warna dalam seluruh aspek desain.

3. Bentuk dasar

Bentuk dasar adalah bentuk-bentuk yang mudah ditemui dalam kehidupan sehari-hari seperti kotak, lingkaran, segitiga dan lain sebagainya.



Gambar 2.9 Bentuk Dasar

4. Tipografi

Tipografi merupakan representasi visual dari sebuah bentuk komunikasi *verbal* dan merupakan *property visual* yang pokok dan efektif. Huruf memainkan peranan penting dalam keberhasilan suatu bentuk komunikasi grafis. Dalam media pembelajaran untuk anak-anak, sebuah huruf harus *legible* yaitu jelas dan memiliki tingkat kemudahan untuk dibaca.

5. Audio

Audio adalah sinyal elektrik yang digunakan untuk membawa suara dalam batas pendengaran manusia. *Audio* merupakan komponen sistem yang sudah termasuk didalamnya atau dapat ditambahkan pada komputer.

2.11 Komputer Grafis

Secara sederhana komputer grafis adalah gambar yang dibuat dengan menggunakan komputer [13]. Pada saat ini penggunaan dari komputer grafis sudah banyak dilihat di komputer, televisi, dan media yang lainnya.

2.11.1 Aplikasi Komputer Grafis

Pengembangan dari computer grafis didorong oleh karena kebutuhan dari para pengguna dan kemajuan dalam *hardware* dan *software*. Aplikasi-aplikasi dari komputer grafis banyak dan bervariasi. Aplikasi-aplikasi ini dapat dibagi menjadi 4 bagian utama, yaitu [14]:

A. Display of Information

Teknik grafis yang lama berkembang sebagai perantara untuk menyampaikan informasi di antara orang-orang. Walaupun secara lisan maupun tulisan juga memberikan tujuan yang sama, sistem visual manusia tidak dapat menyaingi suatu *processor* dari data dan *pattern recognizer.* Oleh karena itu, saat ini banyak dibuat suatu alat yang dapat menganalisis data secara cepat dan tepat untuk menghasilkan informasi yang diinginkan. Contoh dari alat tersebut di antaranya adalah alat-alat pencitraan dalam bidang kedokteran seperti: *Computed Tomography* (CT), *Magnetic Resonance Imaging* (MRI), *Ultrasound*, dan *Positron Emission Tomography* (PET).

B. Design

Desain sangat diperlukan dalam pekerjaan yang berhubungan dengan teknik dan arsitektur. Desain yang dibuat dapat digunakan untuk mencari suatu solusi yang terbaik dalam mengatasi suatu masalah yang berhubungan dengan teknik dan arsitektur. Saat ini penggunaan dari alat-alat pengolah grafik yang interaktif dalam *Computer Aided Design* (CAD) meliputi bidang-bidang seperti:

arsitektur dan desain bagian-bagian mekanik dan *Very Large Scale Integrated Circuits* (VLSI). Dalam aplikasi-aplikasi ini, grafik digunakan dalam cara yang berbeda satu sama yang lainnya.

C. Simulation and animation

Saat sistem grafis berkembang menjadi mampu untuk membuat gambar-gambar yang rumit secara *real time*, para insinyur dan para peneliti mulai menggunakan sistem grafis ini sebagai simulator. Salah satu kegunaannya adalah dalam pelatihan para pilot. *Graphical flight simulator* sudah membuktikan dapat meningkatkan keamanan dan untuk mengurangi pengeluaran dalam pelatihan. Kesuksesan dari *flight simulator* ini menjadi pendorong penggunaan komputer grafis untun animasi pada televise, bioskop, dan industri periklanan. Seluruh film yang beranimasi saat ini dapat dibuat oleh komputer dengan harga yang lebih murah daripada film yang dibuat dengan menggunakan cara yang tradisional.

D. User interface

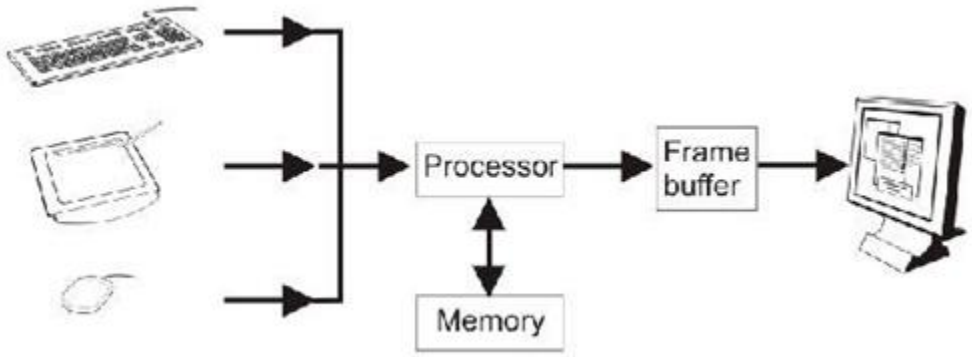
Saat ini, jutaan orang sudah menjadi pengguna dari internet. Akses yang mereka lakukan adalah melalui *graphical network browser*, seperti *Firefox* dan *internet explorer*, yang mempunyai *interface tool* yang sama. Kemajuan dari komputer grafis sudah memungkinkan membuat bentuk *interface* yang bermacam-macam. *Color plate* 14 menunjukkan bahwa *interface* digunakan dengan *high-level modeling package*. Hal ini menunjukkan keanekaragaman tool yang tersedia pada package itu dan karena adanya *device* yang interaktif, pengguna dapat bekerja dalam pemodelan obyek-obyek geometris.

2.11.2 Sistem Grafis

Suatu sistem komputer grafis adalah suatu sistem komputer. Sistem komputer grafis harus mempunyai semua komponen dari sistem komputer. Gambar 2.10 menunjukkan blok diagram dari suatu grafis sistem. Ada 5 elemen utama dalam sistem grafis, yaitu [14]:

1. *Processor*
2. *Memory*
3. *Frame Buffer*
4. *Output devices*

5. Input Devices



Gambar 2.10 Sistem Grafis

Model ini cukup umum untuk memasukkan *workstation*, *personal computer*, *graphic terminal* dihubungkan pada *central time-shared computer* dan *sistem image generation* yang rumit. Walaupun semua komponen dengan pengecualian *frame buffer*, dihadirkan pada komputer grafis yang menggolongkan diagram ini sebagai suatu gambaran dari sistem grafis.

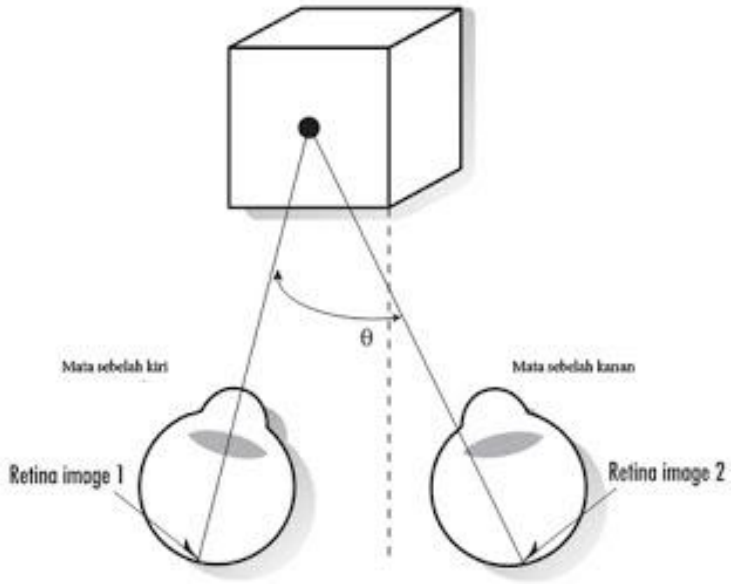
2.11.3 Konsep Dasar Grafis 3 Dimensi

Hal-hal yang akan dibahas meliputi: bagaimana mata merasa bahwa dalam dunia 3 dimensi, bagaimana suatu gambar 2 dimensi dapat memiliki tampilan gambar 3 dimensi, bagaimana koordinat kartesius menentukan letak obyek, bagaimana *viewport* mempengaruhi dimensi gambar, dan bagaimana bekerja dengan proyeksi gambar [15].

A. Penglihatan 3 Dimensi

Komputer grafis 3 dimensi sebetulnya adalah gambar 2 dimensi pada layar komputer yang datar yang menyediakan ilusi dari kedalaman, atau "dimensi" ketiga. Supaya benar-benar melihat dalam 3 dimensi, obyek perlu dilihat dengan kedua mata. Gambar 2.11 memperlihatkan bahwa setiap mata menerima suatu gambar 2 dimensi yang nampak seperti suatu gambar sementara pada retina (bagian belakang dari mata). Kedua gambar ini sedikit berbeda karena kedua mata itu menerima gambar pada 2 sudut yang berbeda. Otak kemudian menggabungkan

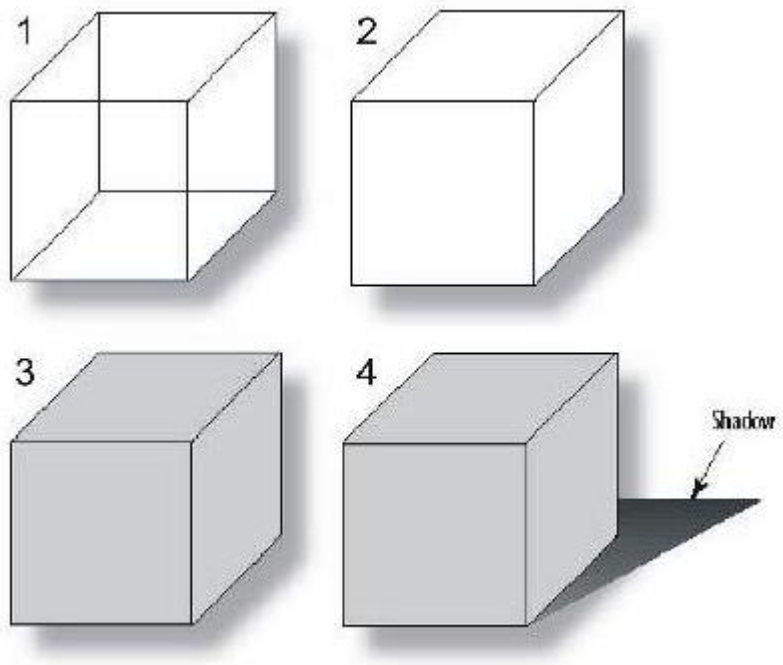
gambar-gambar yang sedikit berbeda ini untuk menghasilkan gambar 3 dimensi yang tunggal di kepala.



Gambar 2.11 Bagaimana mata melihat 3 dimensi

B. Gambar 3 Dimensi dalam Bidang 2 Dimensi

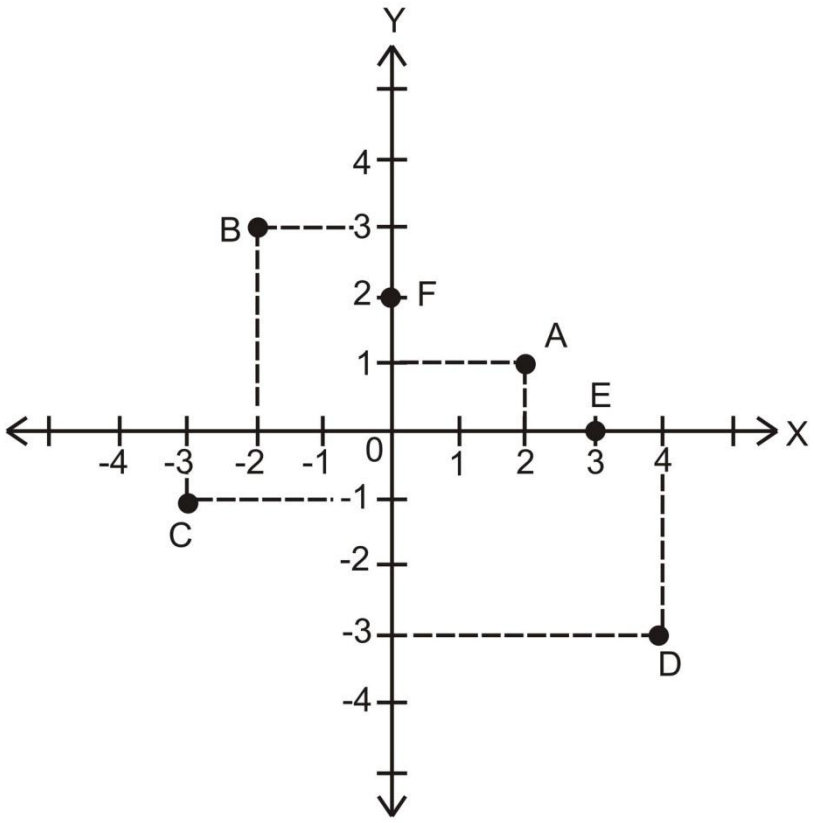
Penyebab yang nyata gambar 3 dimensi dapat dilihat dalam bidang 2 dimensi adalah karena obyek-obyek yang lebih dekat pada mata kelihatan lebih besar daripada obyek-obyek yang lebih jauh dari mata. Efek ini dinamakan perspektif. Perspektif ditambah perubahan warna, tekstur, pencahayaan, pembuatan bayangan, dan variasi dari intensitas warna secara bersama menambahkan pada penglihatan akan gambar 3 dimensi. Perspektif sendiri sudah cukup untuk membuat gambar tampak 3 dimensi. Gambar 2.12 menggambarkan kubus sederhana dengan variasi pembuat gambar nampak kelihatan 3 dimensi. Secara berurutan gambar mulai nomor 1 adalah sebagai berikut: gambar kubus dengan sumua garis pembentuknya, gambar kubus setelah menghilangkan garis yang tidak kelihatan, gambar kubus dengan warna, gambar kubus dengan warna dan bayangan.



Gambar 2.12 Variasi Kubus Sederhana

C. Koordinat Kartesius

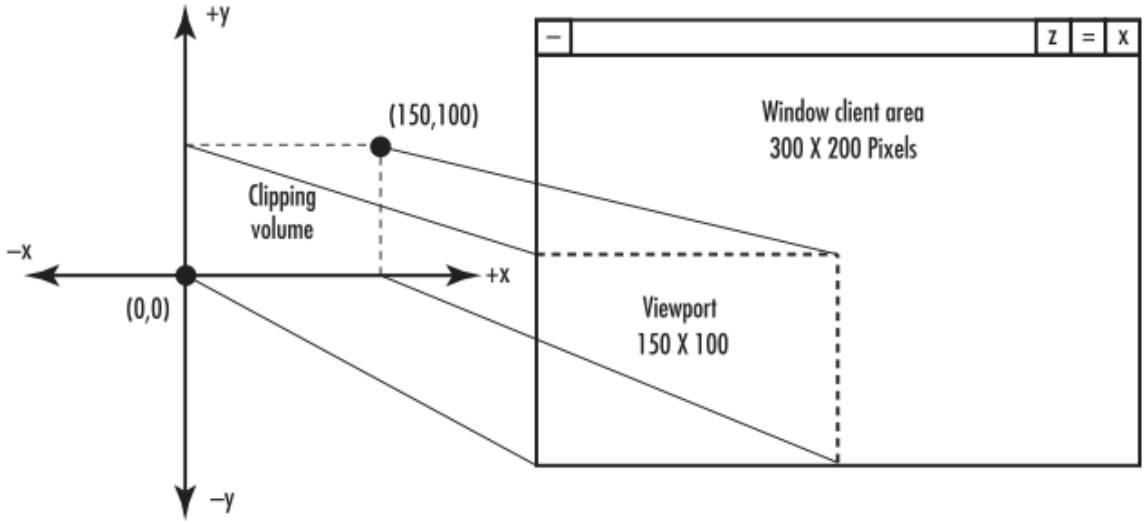
Sistem koordinat yang paling umum untuk penggambaran 2 dimensi adalah sistem koordinat kartesius. Koordinat kartesius ditentukan oleh suatu koordinat x dan suatu koordinat y. Koordinat x adalah suatu ukuran posisi dalam arah horizontal dan y adalah suatu ukuran posisi dalam arah vertical. Koordinat awal dari sistem kartesius adalah pada x=0 dan y=0. Koordinat kartesius ditulis sepasang dalam tanda kurung dengan koordinat x ditulis dahulu, lalu diikuti koordinat y yang dipisahkan dengan tanda koma. Gambar 2.13 sistem koordinat kartesius dalam 2 dimensi. X-axis dan y-axis tegak lurus dan bersama mendefinisikan bidang datar xy.



Gambar 2.13 Koordinat Kartesius

D. Viewport

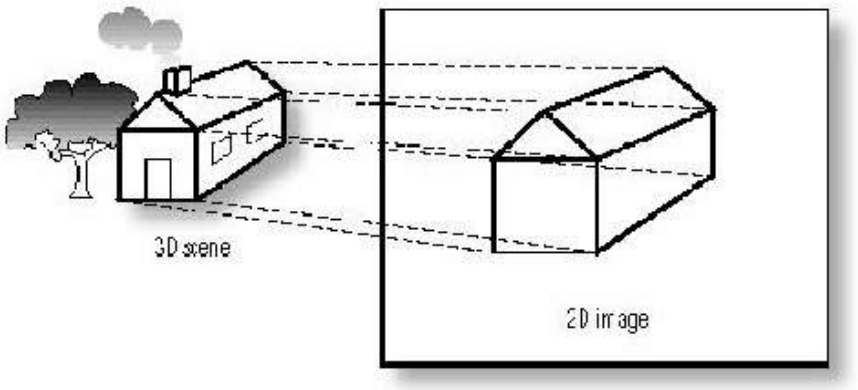
Sistem koordinat harus dipetakan dari *logical Cartesian coordinate* ke *physical screen pixel coordinate*. Pemetaan ini ditentukan oleh suatu pengaturan yang dikenal sebagai *viewport*. *Viewport* adalah daerah di dalam area *window* yang akan digunakan untuk menggambar pada area *clipping*. *Viewport* secara sederhana memetakan area *clipping* ke suatu daerah pada *window*. Biasanya *viewport* didefinisikan sebagai keseluruhan *window*, tapi ini tidak harus, sebagian dari *window* juga dapat digunakan. Gambar 2.14 menunjukkan *window* yang berukuran 300 x 200 *pixel* dengan *viewport* yang didefinisikan sebagai keseluruhan area *window*. Bila area *clipping* untuk window ini diatur menjadi 0 sampai 150 sepanjang x-axis dan 0 sampai 100 sepanjang y-axis, kemudian *logical coordinate* akan dipetakan ke sistem koordinat layar yang lebih besar dalam *window*. Setiap penambahan dalam sistem *logical coordinate* akan dicocokkan oleh 2 penambahan dalam sistem *physical coordinate* dari *window*.



Gambar 2.14 *Viewport*

E. Proyeksi Gambar

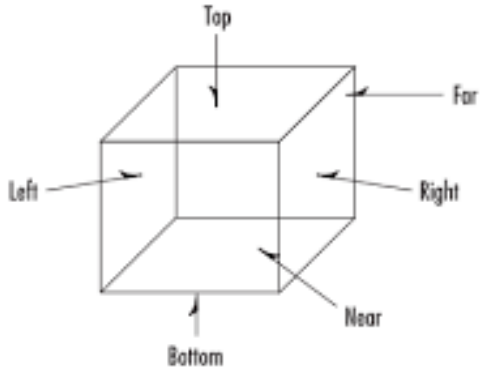
Ada 2 metode dasar untuk memproyeksikan obyek 3 dimensi menjadi tampilan permukaan 2 dimensi. Semua titik dari obyek dapat diproyeksikan pada permukaan sepanjang garis paralel, atau titik-titik dapat diproyeksikan sepanjang garis yang saling bertemu pada suatu posisi yang dinamakan pusat dari proyeksi. Dua metode proyeksi itu adalah proyeksi pararel dan proyeksi perspektif.



Gambar 2.15 Proyeksi Gambar 3D menjadi gambar 2D

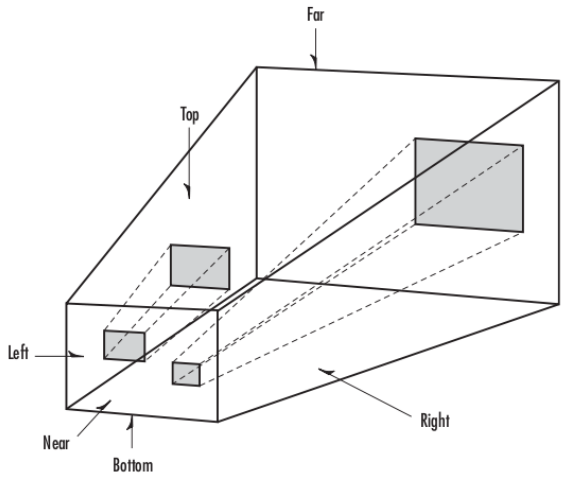
Proyeksi yang pertama adalah proyeksi pararel seperti yang ditunjukkan pada gambar 2.16. Proyeksi ini digunakan dengan menentukan *clipping volume* dalam bentuk empat persegi panjang. Apa saja di luar area *clipping* ini tidak digambar. *Clipping volume* dalam proyeksi paralel ditentukan dengan menentukan jauh, dekat, kiri, kanan, atas, dan bawah dari bidang *clipping*. Obyek dan gambar

yang diletakkan di dalam *clipping volume* kemudian diproyeksikan menjadi gambar 2 dimensi yang nampak pada layar.



Gambar 2.16 *Clipping volume* untuk proyeksi pararel

Proyeksi yang kedua adalah proyeksi perspektif seperti yang ditunjukkan pada gambar 2.17. Proyeksi ini membuat obyek-obyek yang jauh kelihatan lebih kecil dibandingkan obyek-obyek yang lebih dekat. *Clipping volume* dari proyeksi ini seperti suatu *pyramid* dengan puncak yang dipotong. Potongan puncak ini dinamakan *frustum*. Obyek-obyek yang lebih dekat dengan bagian depan dari *clipping volume* nampak mendekati ukuran aslinya, sementara obyek-obyek yang mendekati bagian belakan dari *clipping volume* menyusut karena obyek-obyek itu diproyeksikan ke depat dari *clipping volume*. Tipe proyeksi ini memberi gambaran yang paling nyata untuk simulasi dan 3 dimensi.



Gambar 2.17 *Clipping volume* untuk proyeksi perspektif

2.12 OOP (*Object Oriented Programming*)

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem internet banking, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika kita menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu [8]:

1. Alami (*Natural*).
2. Dapat diandalkan (*Reliable*).
3. Dapat digunakan kembali (*Reusable*).
4. Mudah untuk dalam perawatan (*Maintainable*).
5. Dapat diperluas (*Extendable*).
6. Efisiensi waktu.

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep OOP, adalah :

1. *C++*.
2. *Visual C++*.
3. *Visual Basic*.
4. *Java*.

2.13 UML (*Unified Modelling Language*)

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sebuah sistem. *UML* adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. *UML* adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-

an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

UML mendefinisikan diagram-diagram sebagai berikut [13]:

1). *use case diagram*
2). *class diagram*
3). *statechart diagram*
4). *activity diagram*
5). *sequence diagram*
6). *collaboration diagram*
7). *component diagram*
8). *deployment diagram*

2.13.1 Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

2.13.2 Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

2.13.3 Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring.

2.13.4 Activity Diagram

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state*

sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

2.13.5 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pemain, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

2.13.6 Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram,* tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number,* di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari *level* yang sama memiliki *prefiks* yang sama

2.13.7 Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time, link time*, maupun *run time*.

Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga

berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

2.13.8 Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, *server* atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server*, dan hal-hal lain yang bersifat fisikal

Sebuah *node* adalah *server*, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

2.14 Unity

Unity adalah sebuah sebuah *tool* yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. Unity bisa untuk *game*s PC dan *games Online*. Untuk *games Online* kita memerlukan sebuah *plugin*, yaitu *Unity Web Player*, sama halnya dengan *Flash Player* pada *Browser*. Bahasa pemrograman yang digunakan bermacam-macam, mulai dari *Javascript*, C#, dan *Boo* .

Pada unity, kita tidak bisa melakukan desain / *modelling*, dikarenakan unity bukan *tool* untuk mendesain Jadi jika kita ingin mendesain, kita memerlukan 3D editor lain seperti *3dsmax* atau *Blender*. Banyak hal yang bisa di lakukan di unity , ada fitur *audio reverb zone* , *particle effect* , *Sky Box* Untuk menambahkan langit , dan masih banyak lagi tentunya .kita juga bisa langsung mengedit *texture* dari *editor* seperti *photoshop* dll , unity bagus untuk pemula maupun *expert* [14] .

Features (Scripting)

1. Mendukung 3 bahasa pemrograman, *JavaScript*, C#, dan Boo.
2. *Flexible and EasyMoving*, *rotating*, dan *scaling objects* hanya perlu sebaris kode . Begitu juga dengan *Duplicating*, *removing*, dan *changing properties*.
3. *Multi Platform Game* bisa di deploy di PC, Mac, Wii, iPhone, iPad dan *browser*, android.

4. *Visual Properties Variables* yang di definisikan dengan *scripts* ditampilkan pada Editor. Bisa digeser, di *drag* and *drop*, bisa memilih warna dengan *color picker*
5. Berbasis .NET –>Penjalanan program dilakukan dengan *Open Source* .NET *platform, Mono.*