Bab 2

Output Primitif

**TUJUAN PEMBELAJARAN**

* Agar pembaca mengetahui definisi output primitif.
* Agar pembaca memahami primitif geometri titik dan garis.
* Agar pembaca memahami algoritma pembentukan garis *brute force*, DDA (*Digital Diferential Analyzer* )dan Bressenham.
* Agar pembaca memahami algoritma pembentukan lingkaran
* Agar pembaca memahami algoritma *Area Filling scan line* dan *Boundary Fill*

**OUTCOME PEMBELAJARAN**

* Pembaca dapat menjelaskan dan memberikan contoh tentang definisi output primitif.
* Pembaca dapat menjelaskan primitif geometri titik dan garis.
* Pembaca dapat menjelaskan perbedaan algoritma pembentukan garis brute force, DDA dan Bressenham.
* Pembaca dapat menjelaskan algoritma pembentukan lingkaran
* Pembaca dapat menjelaskan perbedaan algoritma *Area Filling scan line* dan *Boundary Fill*

**Pendahuluan**

Output primitif adalah struktur dasar geometri yang paling sederhana dari gambar grafika komputer. Titik dan garis adalah contoh dari output primitif yang dapat digunakan untuk membentuk gambar, misalnya lingkaran, kerucut, permukaan berbentuk persegi, kurva dan permukaan berbentuk lengkung, warna area dan karakter, dan lain-lain. Gambar-gambar seperti rumah, tembok, gedung dan lain-lain merupakan obyek yang lengkap yang ditempatkan pada koordinat tertentu pada layar, dimana obyek-obyek tadi sebenarnya tersusun dari kumpulan titik dan garis sebagai penyusunnya.

**2.1 *Piksel* (*Picture Element*)**

*Piksel* adalah elemen gambar terkecil berupa sebuah titik yang ditempatkan dilayar.

Jumlah kolom *piksel* = *N*

Sistem Koordinat

(0,0)

*x*

*y*

Piksel berada di posisi (3,2)

1 2 3 *N*

1

2

3

*M*

Jumlah baris *piksel* = *M*

Gambar 2.1: Layar berisi M baris *piksel* dan N kolom *piksel.* Sebuah piksel terletak diposisi (3,2).

Gambar 2.1 menunjukkan sebuah piksel mempunyai koordinat (3,2) pada layar. Jumlah Piksel untuk setiap gambar tergantung dari kemampuan *Graphics card.* Terdapat beberapa tampilan format gambar:

|  |  |  |  |
| --- | --- | --- | --- |
| Standart | x-maksimal | y-maksimal | Jumlah keseluruhan Piksel |
| VGA  SVGA  XGA  SXGA | 640  800  1024  1280 | 480  600  768  1024 | 307 200  480 000  786 432  1 228 800 |

Sebagai pembanding adalah tayangan televisi standart dengan jumlah titik gambar sebesar 400.000 Piksel. Bandingkan dengan WebCam standar VGA dan juga dengan HP yang memiliki fasilitas perekam 2 Mega Piksel.

**Resolusi Piksel** adalah jumlah piksel persatuan panjang.

Jika layar mempunyai dimensi ***p***inchi sepanjang *x* dan ***q*** inchi sepanjang *y,* maka :

resolusi-x =  piksel/inchi (ppi) (2-1)

resolusi-y =  piksel/inchi (ppi) (2-2)

Aspek rasio =  (2-3)

***Frame buffer* adalah**  area memory tempat informasi gambar disimpan.Jika system mempunyai *n* bit memori untuk tiap-tiap piksel, maka ukuran *frame buffer*adalah = MN*n*/8 byte. Jumlah warna yang bisa ditampilkan secara bersama-sama dilayar adalah 2*n*. Sistem dikatakan mempunyai *n* ***bit planes***.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Screen Mode | Piksels | Screen Ratio | Bit planes | Colors | Memory |
| CGA(med res) | 320x200 | 4:3 | 2 | 4 | 16K |
| CGA(high res) | 640x200 | 4:3 | 1 | 1 | 16K |
| EGA | 640x350 | 4:3 | 4 | 16 | 112K |
| VGA | 640x480 | 4:3 | 8 | 256 | 307K |
| Super VGA-1 | 800x600 | 4:3 | 24 | 16.7M | 1.5M |
| Super VGA-2 | 1024x768 | 4:3 | 16 | 65K | 1.6M |

Atribut dari piksel adalah warnanya (indeks warna) *i*, dimana 0 ≤ *i* ≤ 2*n*. Biasanya indeks warna yang bernilai 0 melambangkan warna hitam.

Monitor monokrom yang mempunyai satu *bit plane* hanya bisa menampilkan dua warna, yaitu warna hitam (0) sebagai backgrounddan warna putih (1)sebagai foreground. Monitor monokrom yang mempunyai 8 *bit plane* bisa menampilkan 256 warna yang berbeda, yang disebut dengan grayscale dengan indeks warna dimulai dari 0 (hitam) sampai dengan 255 (putih).

Contoh 2.1

Diketahui sebuah sistem dengan 1280 X 1024 frame buffer, mempunyai 4 bits per piksel. jika sistem tersebut dapat mentransfer data 104 bits per detik, berapa lama waktu yang dibutuhkan untuk loading ?

Jawab:

Jumlah piksel pada frame buffer = 1280 x 1024 = 1310720 piksel

Jumlah bit pada frame buffer = 1310720 piksel x 4 bit /piksel = 5242880 bit

Waktu loadingnya = 

**2.2 Warna**

Sudah dikatakan diatas bahwa warna merupakan atribut dari piksel. Selain piksel, yang lebih penting diperhitungkan lagi adalah warna dari piksel itu sendiri. Setiap piksel mengandung informasi mengenai warna dengan jumlah yang beragam. Pada awalnya saat teknik penggambaran belum begitu maju, hanya disediakan 16 kemungkinan warna setiap Pikselnya. Tetapi *Graphics card* sekarang ini sudah mampu menampilkan gambar dengan berbagai macam kedalaman seperti pada tabel dibawah ini:

|  |  |  |
| --- | --- | --- |
| Jumlah Byte setiap Piksel | Jumlah tampilan warna | Color Quality |
| 1 (8 bit)  2 (16 bit)  3 (24 bit)  4 (32 bit) | 256  65 536  16 777 216  4 294 967 296 | Low  Medium  High  Highest |

Untuk menampilkan suatu gambar pada layar monitor sering kali tidak cukup hanya dengan 256 tampilan warna. Tetapi jika jumlah tampilan warna diperbesar maka memerlukan ruang (memori) lebih banyak lagi. Lihat tabel kebutuhan memori di bawah ini:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x/y-Format | 1 B / Piksel | 2 B / Piksel | 3 B / Piksel | 4 B / Piksel |
| 640 x 480  800 x 600  1024 x 768  1280 x 1024 | 300 kB  469 kB  768 kB  1311 kB | 600 kB  938 kB  1536 kB  2622 kB | 900 kB  1406 kB  2304 kB  3933 kB | 1200 kB  1876 kB  3072 kB  5244 kB |

Untuk menampilkan gambar pada layar monitor tabung sehingga tanpa efek kerdipan (*flicker-free*)diperlukan pengulangan tayang sebanyak 75 kali dalam satu detik. Bisa dicoba pada layar monitor, turunkan frekuensi monitor anda pada penyetelan: display – settings – advanced – Monitor – Screen refresh rate = 60. Terasa kerdipan yang mengganggu dan tidak nyaman di mata. Pengulangan tayangan per detik yang begitu tinggi (lebih besar dari 75) pada resolusi (ketajaman gambar) yang tinggi pula, dapat menyebabkan kesulitan dalam memindahkan gambar dari *graphic card* ke layar monitor (diperlukan *transfer-rate* yang besar). Berbeda dengan layer LCD (Liquid Crystal Display). Pada layer LCD tidak dijumpai efek kerdipan (*flicker*) sehingga tidak memerlukan frekuensi pengulangan gambar yang tinggi. Hanya saja, yang masih menjadi masalah pada layar LCD adalah sudut pandang yang terbatas, *delay time*, dan tingginya harga. Secara teknis warna pada pesawat televisi tidak dipindahkan dalam bentuk Byte untuk setiap elemen gambarnya, akan tetapi sudah berupa sinyal warna analog. Sebagai pembanding *Photo-CD* yang dikeluarkan oleh Kodak memiliki resolusi sebesar 2500 x 3000 Piksel dengan kedalaman 3 Byte warna. Perbandingan ke tiga format dapat dilihat pada tabel dibawah ini:

|  |  |  |
| --- | --- | --- |
| Televisi standar | Gambar komputer  (600 x 800 x 3 Byte) | Photo-CD |
| 800 kilo Byte | 1.5 Mega Byte | 20 Mega Byte |

**2.3 Garis**

Software untuk menggambar garis lurus pada setiap paket grafika komputer secara tradisi sebagai fungsi elementarnya sudah tersedia, tetapi ada baiknya kita simak lebih seksama bagaimana algoritma sebagai dasar penggambaran titik-titik pada *raster graphics*. *Raster graphics* diartikan sebagai grafik yang dibangun atas dasar titik-titik pada kolom dan baris yang juga mengandung informasi warna. Gambar 2.2 adalah sebuah garis lurus yang melalui titik (*x*1*,y*1) dan (*x*2*,y*2)*.*

P1(*x*1,*y*1)

P2(*x*2,*y*2)

Gambar 2.2: Garis lurus yang melalui titik P1(*x*1*,y*1) dan P2(*x*2*,y*2)

Secara umum garis lurus dinyatakan dalam persamaan :

**y = m*x* + c** (2-4)

dimana : m adalah gradient dan **c** adalah konstanta.

Bila garis melalui titik (*x*1, *y*1) dan titik (*x*2, *y*2) maka gradien ***m*** dihitung menggunakan persamaan (2-5)



(2-5)

Persamaan garis bergradien ***m***  yang melewati titik (*x*1, *y*1) adalah



(2.6)

Berdasarkan persamaan (2-4), garis yang melalui titik (*x*1, *y*1) dan titik (*x*2, *y*2) dapat ditulis sebagai berikut



atau



atau



(2-7)

bila ∆*x* = 1, maka persamaan (2-7) menjadi

(2-8)



Algoritma untuk menggambar garis pada komputer didasarkan pada persamaan (2-6) dan (2-8).

Tugas dari algoritma pembentuk garis adalah menentukan titik-titik diantara dua titik ujung (*x*1,*y*1) dan (*x*2,*y*2) yang akan digambar sebagai garis.

Kriteria algoritma pembentuk garis yang baik:

1. Garis antara dua titik ujung tersebut sebisa mungkin harus lurus

2. Kerapatan titik-titik konstan (tidak ada *gap* antara dua titik yang bersebelahan)

3. Kerapatan titik-titik tidak tergantung pada kemiringan garis

4. Waktu proses algoritma harus cepat

**2.3.1 Algoritma** **Brute Force**

Algoritma brute force untuk membentuk garis didasarkan pada persamaan (2-6), yaitu :

1. Tentukan dua titik ujung (*x*1,*y*1) dan (*x*2,*y*2)

2. Jika *x*1 = *x*2 (garis vertikal), maka

(a) *y* = *y*+ 1 dan *x* tetap

(b) gambar titik (*x,y*) di layar

(c) Selesai

3. Jika *y*1 = *y*2 (garis horisontal), maka

(a) *x* = *x*+ 1 dan *y* tetap

(b) gambar titik (*x,y*) di layar

(c) Selesai

{anggap *x*2> *x*1 , (jika sebaliknya, gantilah *x*2dengan *x*1 )}

4. Hitung kemiringan garis *m* = (*y*2 − *y*1)/( *x*2− *x*1)

5. N = *x*2− *x*1 +1

6. *x* = *x*1

7. Ulang sebanyak N kali:

(a)



(b) lakukan pembulatan *ya* = Round(*y*)*,*

(c) gambar titik (*x,ya*) di layar

(d) *x* = *x* + 1

8. selesai

Perhatikan langkah 7(b), karena posisi piksel adalah integer (bilangan bulat), dan nilai gradien *m* biasanya pecahan, maka nilai *y* adalah bilangan pecahan, sehingga kita perlu melakukan pembulatan terhadap nilai *y*. Perhatikan Gambar 2.3 berikut.

Garis yang dicari

**(xi+1,Round(yi+m))**

**(xi+1, yi+m)**

**(xi, yi)**

**(xi, Round(yi))**

Gambar 2.3: Pembulatan nilai *y* dilakukan karena posisi piksel adalah bulat (integer)

Contoh 2.2

Diketahui 2 buah titik A(2,1) dan titik B(8,5) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma *Brute Force*.

Jawab:

1. titik ujung (*x*1,*y*1) = (2,1) dan (*x*2,*y*2) = (8,5)
2. tidak dipenuhi
3. tidak dipenuhi
4. *m* = (5 – 1)/(8 – 2) = 0,67
5. N = 8 – 2 + 1 = 7
6. untuk *x* = 2 *y* = 0,67(*x* – 2) + 1
7. Ulang sebanyak 7 kali

iterasi ke-1:

*x* = 2

*y* = 0,67. (2 – 2) + 1 = 1

pembulatan: *y* = 1

gambar titik (2,1) di layar .

*x* = 2 + 1 = 3

iterasi ke-2:

*x* = 3

*y* = 0,67. (3 – 2) + 1 = 1,67

pembulatan: *y* = 2

gambar titik (3,2) di layar .

*x* = 3 + 1 = 4

iterasi ke-3:

*x* = 4

*y* = 0,67. (4 – 2) + 1 = 2,34

pembulatan: *y* = 2

gambar titik (4,2) di layar .

*x* = 4 + 1 = 5

iterasi ke-4:

*x* = 5

*y* = 0,67. (5 – 2) + 1 = 3,01

pembulatan: *y* = 3

gambar titik (5,3) di layar .

*x* = 5 + 1 = 6

iterasi ke-5:

*x* = 6

*y* = 0,67. (3 – 2) + 1 = 3,68

pembulatan: *y* = 4

gambar titik (6,4) di layar .

*x* = 6 + 1 = 7

iterasi ke-6:

*x* = 7

*y* = 0,67. (7 – 2) + 1 =4,35

pembulatan: *y* = 4

gambar titik (7,4) di layar .

*x* = 7 + 1 = 8

iterasi ke-7:

*x* = 8

*y* = 0,67. (7 – 2) + 1 =5,02

pembulatan: *y* = 5

gambar titik (8,5) di layar .

titik-titik pembentuk garis : (2,1), (3,2), (4,2), (5,3), (6,4), (7,4), (8,5).

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

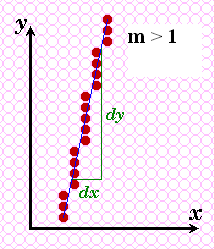
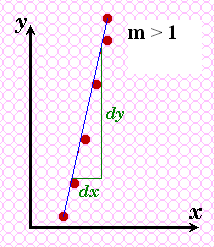
22

1

Gambar 2.4: Titik-titik pembentuk garis hasil perhitungan menggunakan algoritma *Brute Force* digambar pada *raster graphics.*

**Masalah :**

Untuk kemiringan ***m*** > 1, garis menjadi tidak kontinyu (Gambar 2.5(a)) yang mengakibatkan terjadinya *gap* antar piksel, sehingga diperlukan interpolasi (Gambar 2.5(b)).



(a)

(b)

Gambar 2.5: (a) Garis dengan kemiringan ***m*** > 1, tampak bahwa garis tidak kontinyu (b) setelah dilakukan interpolasi garis menjadi kontinyu.

Selain menggunakan interpolasi, untuk kemiringan garis ***m*** > 1, tukarlah *x* dengan *y*  maka sudah tidak terjadi *gap* antara titik yang satu dengan yang lain. Sehingga algoritma pembentukan garis untuk ***m*** > 1 adalah sebagai berikut :

1. Tentukan dua titik ujung (*x*1,y1) dan (*x*2,*y*2)

2. Jika *x*1 = *x*2 (garis vertikal), maka

(a) *y* = *y*+ 1 dan *x* tetap

(b) gambar titik (*x,y*) di layar

(c) Selesai

3. Jika *y*1 = *y*2 (garis horisontal), maka

(a) *x* = *x*+ 1 dan *y* tetap

(b) gambar titik (*x,y*) di layar

(c) Selesai

{anggap *y*2> *y*1 , (jika sebaliknya, gantilah *y*2dengan *y*1 )}

4. Hitung kemiringan garis *m* = ( *x*2− *x*1) /(*y*2 − *y*1)

5. N = *y*2− *y*1+1

6. *y* = *y*1

7. Ulang sebanyak N kali:



(a)

(b) lakukan pembulatan *xa* = Round(*x*)*,*

(c) gambar titik (*xa,y*) di layar

(d) *y* = *y* + 1

8. selesai

Contoh 2.3

Diketahui 2 buah titik A(4,3) dan titik B(7,8) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma *Brute Force*.

Jawab:

1. titik ujung A(4,3) dan B(7,8)
2. tidak dipenuhi
3. tidak dipenuhi
4. *m* = (7 – 4)/(8 – 3) = 0,6
5. N = 8 – 3 + 1 = 6
6. untuk *y* = 3 dan *x* = 0,6. (*y* – 3) + 4
7. Ulang sebanyak 6 kali

iterasi ke-1:

*y*= 3

*x* = 0,6. (3 – 3) + 4 =4

pembulatan: *x* = 4

gambar titik (4,3) di layar .

*y* = 3+ 1 = 4

iterasi ke-2:

*y*= 4

*x* = 0,6. (4 – 3) + 4 =4,6

pembulatan: *x* = 5

gambar titik (5,4) di layar .

*y* = 4+ 1 = 5

iterasi ke-3:

*y*= 5

*x* = 0,6. (5 – 3) + 4 =5,2

pembulatan: *x* = 5

gambar titik (5,5) di layar .

*y* = 5+ 1 = 6

iterasi ke-4:

*y*= 6

*x* = 0,6. (6 – 3) + 4 =5,8

pembulatan: *x* = 6

gambar titik (6,6) di layar .

*y* = 6+ 1 = 7

iterasi ke-5:

*y*= 7

*x* = 0,6. (7 – 3) + 4 =6,4

pembulatan: *x* = 6

gambar titik (6,7) di layar .

*y* = 7+ 1 = 8

iterasi ke-6:

*y*= 8

*x* = 0,6. (8 – 3) + 4 =7

pembulatan: *x* = 7

gambar titik (7,8) di layar .

*y* = 8+ 1 = 9

titik-titik pembentuk garis : (4,3), (5,4), (5,5), (6,6), (6,7), (7,8).

Kelemahan algoritma *Brute Force* :

Algoritma ini terlalu lambat karena: pada setiap iterasi terdapat perkalian bilangan pecahan (*floating point*), penjumlahan bilangan pecahan, dan proses pembulatan angka pecahan menjadi bulat (integer).

**2.3.2 Algoritma DDA (*Digital Diferential Analyzer* )**

DDA adalah algoritma pembentuk garis yang didasarkan pada perasamaan (2-8). Garis dibuat menggunakan titik awal (*x*1, *y*1) dan titik akhir (*x*2, *y*2). Setiap koordinat titik (*x*k, *y*k) yang membentuk garis diperoleh dari perhitungan, kemudian hasil perhitungan dikonversikan menjadi nilai integer. Berikut adalah langkah-langkah pembentukan garis berdasarkan algoritma DDA:

===================================================================

1. Tentukan dua titik yang akan dihubungkan dalam pembentukan garis.

2. Tentukan salah satunya sebagai titik awal (*x*1, *y*1) dan yang lain sebagai titik akhir (*x*2, *y*2).

3. Hitung : *dx* = *x*2 − *x*1dan *dy* = *y*2− *y*1

4. Tentukan *step,* dengan ketentuan berikut:

- bila |*dx| >* |*dy*| maka *step* = *|dx*|

- bila tidak, maka *step* = |*dy*|

5. Hitung penambahan koordinat piksel dengan persamaan:

*x\_inc* = *dx* / *step*

*y\_inc* = *dy* / *step*

6. Koordinat selanjutnya :

*x* = *x*+ *x\_inc* *y* = *y* + *y\_inc*

7. Lakukan pembulatan *u* = Round(*x*), *v* = Round(*x*), kemudian plot piksel (*u, v*) pada layar

1. Ulangi point 6 dan 7 untuk menentukan posisi piksel berikutnya sampai *x* = *x*2 dan *y* = *y*2.

Contoh 2.4

Diketahui 2 buah titik A(2,1) dan titik B(8,5) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma DDA.

Jawab:

Titik awal (*x*1, *y*1) = A(2,1) dan Titik akhir (*x*2, *y*2) = B(8,5)

*dx* = *x*2 − *x*1 = 8 −2 = 6 dan *dy* = *y*2− *y*1 = 5 − 1 = 4

Karena: |*dx*| > |*dy*|, maka *step* = *|dx*| = 6

*x\_inc* = *dx* / *step* = 6/6 = 1

*y\_inc* = *dy* / *step* = 4/6 = 0,67

=================================================

Iterasi ke-1: (*x,y*) = (2,1)

*x*+*x\_inc* = 2 + 1 = 3

*y*+*y\_inc* = 1 + 0,67 = 1,67

Koordinat selanjutnya : (*x,y*) = (3; 1,67)

Pembulatan (3; 1,67) ≈ (3,2). Gambar titik (3,2) dilayar

=================================================

Iterasi ke-2: (*x,y*) = (3; 1,67)

*x*+*x\_inc* = 3 + 1 = 4

*y*+*y\_inc* = 1,67 + 0,67 = 2,34

Koordinat selanjutnya : (*x,y*) = (4; 2,34)

Pembulatan (4; 2,34) ≈ (4,2). Gambar titik (4,2) dilayar

=================================================

Iterasi ke-3: (*x,y*) = (4; 2,34)

*xA*+*x\_inc* = 4 + 1 = 5

*yA*+*y\_inc* = 2,34 + 0,67= 3,01

Koordinat selanjutnya : (*x,y*) = (5; 3,01)

Pembulatan (5; 3,01) ≈ (5,3). Gambar titik (5,3) dilayar

=================================================

Iterasi ke-4: (*x,y*) = (5; 3,01)

*xA*+*x\_inc* = 5 + 1 = 6

*yA*+*y\_inc* = 3,01 + 0,67 = 3,68

Koordinat selanjutnya : (*x,y*) = (6; 3,68)

Pembulatan (6; 3,68) ≈ (6,4). Gambar titik (6,4) dilayar

=================================================

Iterasi ke-5: (*x,y*) = (6; 3,68)

*xA*+*x\_inc* = 6 + 1 = 7

*yA*+*y\_inc* = 3,68 + 0,67 = 4,35

Koordinat selanjutnya : (*x,y*) = (7; 4,35)

Pembulatan (7; 4,35) ≈ (7,4). Gambar titik (7,4) dilayar

=================================================

Iterasi ke-6: (*x,y*) = (7; 4,35)

*xA*+*x\_inc* = 7 + 1 = 8

*yA*+*y\_inc* = 4,35 + 0,67 = 5,02

Koordinat selanjutnya : (*x,y*) = (8; 5,02)

Pembulatan (8; 5,02) ≈ (8,5). Gambar titik (8,5) dilayar

Karena *x* = *x*2 = 8, maka iterasi dihentikan, sehingga diperoleh titik-titik pembentuk garis sebagai berikut: (2,1), (3,2), (4,2), (5,3), (6,4), (7,4) dan (8,5).

=================================================

Bila digambar pada *raster graphics* diperoleh gambar 2.6:

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

22

1

Gambar 2.6: Titik-titik pembentuk garis hasil perhitungan menggunakan algoritma DDA digambar pada *raster graphics.*

**Kelebihan Algoritma DDA dibanding Algoritma *Brute Force***

Algoritma DDA lebih cepat dibanding dengan algoritma *Brute Force* dan baik digunakan untuk kemiringan garis ***m*** > 1.

**Kelemahan Algoritma DDA**

* Prosedur untuk menggambar garis masih menggunakan fungsi pembulatan *x* maupun *y*, sehingga memerlukan waktu.
* variabel *x*, *y* maupun *m* memerlukan bilangan real karena kemiringan merupakan nilai pecahan.

**2.3.3 Algoritma Bressenham**

J.E. Bressenham pada th.1965 mengumumkan penemuan algoritma untuk mengoptimalkan gambar garis pada *raster*. Algoritma garis Bressenham disebut juga Midpoint Line Algorithm.

Algoritma temuannya tidak lagi menggunakan *floating point arihtmetic,* sehingga tidak perlu membulatkan nilai posisi piksel setiap waktudan dalam pengulangan (*looping)* hanya menggunakan pengoperasian incremental. Algoritma ini hampir dapat dipergunakan dan diimplementasikan pada semua hardware dan software untuk keperluan penggambaran garis. Algoritma Bressenham dapat memperbaiki pengendalian plotter, sehingga dia mendapatkan hak paten. Algoritma ini hanya berlaku untuk nilai kemiringan garis : 0 < ***m*** < 1.

Perhatikan Gambar 2.7. Misalkan kemiringan garis ***m****,* yang besarnya adalah 0 < ***m*** < 1, titik awal garis di (*x*0*, y*0) dan titik akhir garis di (*x*1*, y*1). Anggap posisi titik sekarang (*x*p*, y*p) dan sekarang kita harus memilih antara titik E (*East*) atau NE (*Northeast*). Misalkan Q adalah titik potong garis dengan garis *x**= x*p+1, dan M adalah titik tengah antara E dan NE. Jika titik tengah M terletak diatas garis, E dipilih, tetapi jika titik tengah M terletak dibawah garis, NE dipilih.

M

E

NE

(***x*p, *y*p**)

**x =*x*p +1**

Garis aktual

Gambar 2.7: Garis mempunyai kemiringan 0 < ***m*** < 1, pada titik *x**= x*p+1, garis berada diantara dua titik E dan NE yang mempunyai titik tengah di M.

Dalam hal ini kita harus menentukan apakah garis berada diatas titik tengah M atau dibawah titik tengah M. Untuk melakukan ini, kita amati bahwa setiap garis bisa dinyatakan sebagai berikut

F(*x, y*) = *ax + by + c* = 0

Hitung: d*y* = *y*1 – *y*0 dan d*x* = *x*1 – *x*0, maka garis juga bisa ditulis sebagai

*y* = m*x* + B = (d*y*/d*x*) *x* + B

atau

(d*y*/d*x*) *x* – *y* + B = 0

Kalikan dengan d*x* menjadi

(d*y*)*x* – (d*x*)*y* + (d*x*)B = 0

Diperoleh ***a*** = d*y*, ***b*** = −d*x* dan ***c*** = (d*x*)B.

* Jika F(*x, y*) = 0, maka (*x, y*) terletak pada garis
* Jika F(*x, y*) > 0, maka (*x, y*) terletak dibawah garis
* jika F(*x, y*) < 0, maka (*x, y*) terletak diatas garis

Untuk menerapkan kriteria midpoint, kita hitung:

F(M) = F(*x*p + 1, *y*p + ½) = *a* (*x*p + 1) + *b* (*y*p + ½) + *c*

Dan menguji tanda dari ***d*** = F(M).

* Jika ***d*** >0, M terletak dibawah garis, maka NE dipilih.
* Jika ***d*** = 0, pilih E ( sebenarnya memilih NE juga bisa)
* Jika ***d*** < 0, M terletak diatas garis, maka E dipilih.

Misalkan E dipilih, M bertambah satu langkah dalam arah *x*, dan kita mempunyai ***d*** baru misalnya dbaru.

***d***baru = F(*x*p + 2, *y*p + ½) = *a* (*x*p + 2) + *b* (*y*p + ½) + *c*

***d***lama = F(*x*p + 1, *y*p + ½) = *a* (*x*p + 1) + *b* (*y*p + ½) + *c*

jadi ***d***baru –***d***lama = *a* = *dy*.

Jadi setelah E dipilih, pertambahan untuk mendapatkan *d* yang baru adalah *a* = *dy*.

Misalkan NE dipilih, M adalah pertambahan satu langkah kearah *x* dan *y* sehingga

*d*baru = F(*x*p + 2, *y*p + 3/2) = *a* (*x*p + 2) + *b* (*y*p +3/2) + *c*

*d*lama = F(*x*p + 1, *y*p + ½) = *a* (*x*p + 1) + *b* (*y*p + ½) + *c*

jadi *d*baru – dlama = *a* + *b* = d*y* – d*x*.

Untuk memulai algoritma, titik awal adalah (*x*p , *y*p ) = (*x*0*, y*0). Untuk menghitung nilai *d* yang pertama untuk memulai algoritma, *midpoint* pertama adalah (*x*0 + 1, *y*0 + ½) dan

F(M) = F(*x*0 + 1, *y*0 + ½) = *a* (*x*0 + 1) + *b* (*y*0 + ½)+ *c*

= *ax*0 + *by*0 + *c* + *a* + *b*/2

= 0 + *a* + *b*/2 karena (*x*0, *y*0) terletak pada garis.

Jadi kita mulai dari *d* = *a* + *b*/2 = *dy* – *dx*/2

Untuk menghilangkan pecahan, definisikan F dengan mengalikan 2, diperoleh

F(*x,y*) = 2 (*ax + by + c*) = 0.

Definisi ini akan menyebabkan nilai *d* menjadi *d* = 2*dy* – *dx*

Dengan begini algoritma *midpoint* Bresenham (untuk kemiringan 0 < m < 1) adalah :

1. Tentukan dua titik yang akan dihubungkan dalam pembentukan garis.

2. Tentukan salah satu sebagai titik awal (*x*0, *y*0) dan titik akhir (*x*1, *y*1).

1. Hitung *dx*, *dy*,2*dy* dan 2*dy* − 2*dx*
2. Hitung parameter : *p*o = 2*dy* − *dx*

5. Untuk setiap xk sepanjang jalur garis, dimulai dengan k = 0

- bila *p*k *<* 0 maka titik selanjutnya adalah:

(*x*k+1, *y*k) dan *p*k+1 = *p*k + *2dy*

- bila tidak, titik selanjutnya adalah:

(*x*k+1, *y*k+1) dan *p*k+1 = *p*k + *2dy – 2dx*

6. Ulangi nomor 5 untuk menentukan posisi piksel berikutnya, sampai

*x* = *x*1 dan *y* = *y*1.

Contoh 2.5

Diketahui 2 buah titik A(2,1) dan titik B(8,5) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma Bressenham.

Jawab:

Titik awal (*x*0, *y*0) = A(2,1) dan Titik akhir (*x*1, *y*1) = B(8,5)

*dx* = *x*1 − *x*0 = 8 −2 = 6 dan *dy* = *y*1− *y*0 = 5 − 1 = 4

2.*dx* = 2.6 = 12 ; 2.*dy* = 2.4 = 8 dan 2*dy* − 2*dx* = 8 − 12 = −4

*p*o=2.*dy* − *dx* = 8 − 6 = 2

=================================================

Iterasi ke-1 ( k = 0):

Titik awal = (2,1)

*P*o = 2 > 0, maka titik selanjutnya adalah

*x* = 2 + 1 = 3 dan *y* = 1 + 1 = 2, koordinat selanjutnya : (3,2)

*p*1 = *p*0 + *2dy – 2dx* = 2 − 4 = −2

=================================================

Iterasi ke-2 ( k = 1):

Titik awal = (3,2)

*P*1 = −2 < 0, maka titik selanjutnya adalah

*x* = 3 + 1 = 4 dan *y* = 2, koordinat selanjutnya : (4,2)

*p*2 = *p*1 + *2dy* = −2 + 8 = 6

=================================================

Iterasi ke-3 ( k = 2):

Titik awal = (4,2)

*P*2 = 6 > 0, maka titik selanjutnya adalah

*x* = 4 + 1 = 5 dan *y* = 2 + 1 = 3, koordinat selanjutnya : (5,3)

*p*3 = *p*2 + *2dy – 2dx* = 6 − 4 = 2

=================================================

Iterasi ke-4 ( k = 3):

Titik awal = (5,3)

*P*3 = 2 > 0, maka titik selanjutnya adalah

*x* = 5 + 1 = 6 dan *y* = 3 + 1 = 4, koordinat selanjutnya : (6,4)

*p*4 = *p*3 + *2dy – 2dx* = 2 − 4 = −2

=================================================

Iterasi ke-5 ( k = 4):

Titik awal = (6,4)

*P*3 = −2 < 0, maka titik selanjutnya adalah

*x* = 6 + 1 = 7 dan *y* = 4, koordinat selanjutnya : (7,4)

*p*5 = *p*4 + *2dy* = −2 + 8 = 6

=================================================

Iterasi ke-6 ( k = 5):

Titik awal = (7,4)

*P*5 = 6 > 0, maka titik selanjutnya adalah

*x* = 7 + 1 = 8 dan *y* = 4 + 1 = 5, koordinat selanjutnya : (8,5)

Karena *x* = *x*2 = 8, maka iterasi dihentikan, sehingga diperoleh titik-titik penyusun garis sebagai berikut: (2,1), (3,2), (4,2), (5,3), (6,4), (7,4) dan (8,5).

=================================================

Bila digambar pada *raster graphics* diperoleh Gambar 2.8:

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

22

1

Gambar 2.8: Titik-titik pembentuk garis hasil perhitungan menggunakan algoritma Bressenham digambar pada *raster graphics.*

Untuk kemiringan −1 < m < 0 , kita tinggal mengganti komponen berikut:

Untuk setiap *x*k sepanjang jalur garis, dimulai dengan k = 0

- bila *p*k *<* 0 maka titik selanjutnya adalah:

(*x*k+1, *y*k) dan *p*k+1 = *p*k + *2dy*

- bila tidak, titik selanjutnya adalah:

(*x*k+1, *y*k−1) dan *p*k+1 = *p*k + *2dy – 2dx*

Untuk kemiringan garis ***m*** > 1, tukarlah *x* dengan *y*, sehingga algoritma pembentukan garis untuk ***m*** > 1 adalah sebagai berikut :

1. Tentukan dua titik yang akan dihubungkan dalam pembentukan garis.

2. Tentukan salah satu sebagai titik awal (*x*0, *y*0) dan titik akhir (*x*1, *y*1).

1. Hitung *dx*, *dy*,2*dx* dan 2*dx* − 2*dy*
2. Hitung parameter : *p*o = 2*dx* − *dy*

5. Untuk setiap yk sepanjang jalur garis, dimulai dengan k = 0

- bila *p*k *<* 0 maka titik selanjutnya adalah:

(*x*k, *y*k+1) dan *p*k+1 = *p*k + *2dx*

- bila tidak, titik selanjutnya adalah:

(*x*k+1, *y*k+1) dan *p*k+1 = *p*k + *2dx – 2dy*

6. Ulangi nomor 5 untuk menentukan posisi piksel berikutnya, sampai

*x* = *x*1 dan *y* = *y*1.

Contoh 2.6

Diketahui 2 buah titik A(2,1) dan titik B(4,7) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma Bressenham.

Jawab:

Titik awal (*x*0, *y*0) = A(2,1) dan Titik akhir (*x*1, *y*1) = B(4,7)

*dx* = *x*1 − *x*0 = 4 −2 = 2 dan *dy* = *y*1− *y*0 = 7 − 1 = 6

2.*dx* = 4 ; 2.*dy* = 12 dan 2*dx* − 2*dy* = 4 − 12 = −8

*p*o=2.*dx* − *dy* = 4 − 6 = −2

=================================================

Iterasi ke-1 ( k = 0):

Titik awal = (2,1)

*p*o = −2 < 0, maka titik selanjutnya adalah

*x* = 2 dan *y* = 1 + 1 = 2, koordinat selanjutnya : (2,2)

*p*1 = *p*o + *2dx* = −2 + 4 = 2

=================================================

Iterasi ke-2 ( k = 1):

Titik awal = (2,2)

*p*1 = 2 > 0, maka titik selanjutnya adalah

*x* = 2+1 = 3 dan *y* = 2 + 1 = 3, koordinat selanjutnya : (3,3)

*p*2 = *p*1 + 2*dx* − 2*dy* = 2 − 8 = −6

=================================================

Iterasi ke-3 ( k = 2):

Titik awal = (3,3)

*p*2 = −6 < 0, maka titik selanjutnya adalah

*x* = 3 dan *y* = 3 + 1 = 4, koordinat selanjutnya : (3,4)

*p*3 = *p*2 + 2*dx* − 2*dy* = −6 +4 = −2

=================================================

Iterasi ke-4 ( k = 3):

Titik awal = (3,4)

*p*3 = −2 < 0, maka titik selanjutnya adalah

*x* = 3 dan *y* = 4 + 1 = 5, koordinat selanjutnya : (3,5)

*p*4 = *p*3 + 2*dx* − 2*dy* = −2 +4 = 2

=================================================

Iterasi ke-5 ( k = 4):

Titik awal = (3,5)

*p*4 = 2 > 0, maka titik selanjutnya adalah

*x* = 3+1=4 dan *y* = 5 + 1 = 6, koordinat selanjutnya : (4,6)

*p*5 = *p*4 + 2*dx* − 2*dy* = 2 −8 = −6

=================================================

Iterasi ke-6 ( k = 5):

Titik awal = (4,6)

*p*5 = −6< 0, maka titik selanjutnya adalah

*x* = 4 dan *y* = 6 + 1 = 7, koordinat selanjutnya : (4,7)

*p*6 = *p*5 + 2*dx* − 2*dy* = −6+4 = −2

=================================================

Sampai disini iterasi dihentikan.

**2.4 Lingkaran**

Persamaan umum lingkaran dengan pusat lingkaran (*x*p, *y*p) dan jari-jari ***r***adalah:



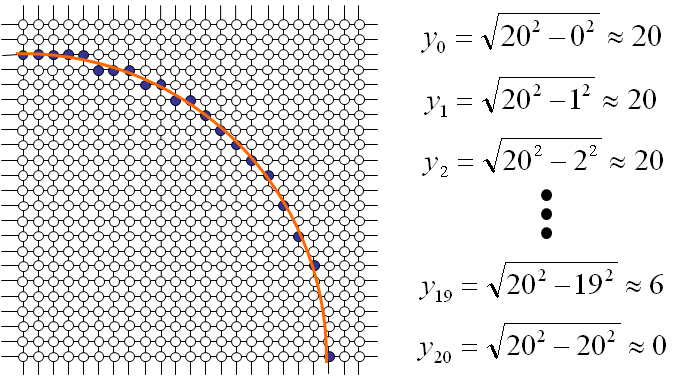
(2-9)

Atau



(2-10)

Untuk menggambar piksel-piksel dalam kurva lingkaran, dapat digunakan sumbu *x* dari *x* = (*x*p-*r*) sampai *x* = (*x*p+*r*) sebagai parameter dan sumbu *y* sebagai hasil dari persamaan (2-10). Algoritma ini memerlukan waktu operasi yang besar, karena mengandung operasi perkalian dua bilangan integer, membutuhkan fungsi SQRT (untuk menghitung akar suatu bilangan) dan fungsi ROUND (untuk membulatkan bilangan pecahan menjadi bilangan integer), dan menghasilkan posisi koordinat piksel yang tidak merata, karena terjadinya gaps yang disebabkan adanya perubahan gradient seperti tampak pada Gambar 2.9.



Gambar 2.9: Terjadinya gaps yang disebabkan adanya perubahan gradient menghasilkan posisi koordinat piksel yang tidak merata

Untuk menghindari posisi koordinat piksel yang tidak merata, koordinat piksel (*x,y*) dinyatakan dengan menggunakan koordinat polar dalam persamaan (2-11)

 dan  (2-11)

Akan tetapi penggambaran lingkaran menggunakan persamaan (2-11) memerlukan waktu operasi yang besar karena mengandung operasi perkalian bilangan riel, perhitungan trigonometri, dan membutuhkan banyak segmen garis.

2.4.1 **Simetris Delapan Titik**

Pembuatan kurva lingkaran dapat dilakukan dengan menentukan titik awal (*x,y*) yang terletak pada lingkaran, maka tujuh titik yang lain (yang terletak pada lingkaran juga) dapat ditentukan sebagai berikut :

(−*x,y*), (*x,* −*y*), (−*x,* −*y*), (*y,x*), (−*y,x*), (*y,* −*x*), (−*y,* −*x*)

Sehingga terbentuk delapan titik:

(*x, y*), (−*x,y*), (*x,* −*y*), (−*x,* −*y*), (*y,x*), (−*y,x*), (*y,* −*x*), (−*y,* −*x*)

Dengan demikian sebenarnya hanya diperlukan untuk menghitung segmen 45° dalam menentukan lingkaran selengkapnya.

(

x,y)

(

y

,x)

(

y,-x)

(

x,-y)

(-

x,-y)

(-

y,-x)

(-

y

,x)

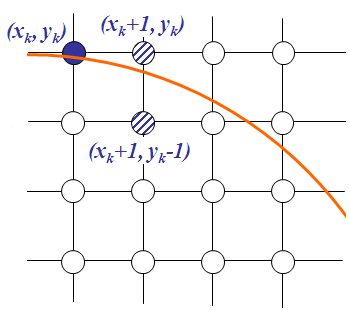
(-

x,y)

Gambar 2.10: Delapan titik simetris pada lingkaran

2.4.2 **Algoritma *Midpoint***

Algoritma *midpoint* juga disebut algoritma lingkaran Bressenham. Pembentukan semua titik berdasarkan titik pusat dengan penambahan semua jalur di sekeliling lingkaran. Komputasi untuk membuat kurva lingkaran dimulai dengan mengidentifikasi bagian-bagian dari lingkaran yang dapat ditentukan dengan menggunakan sifat simetri, hal ini dilakukan dengan cara membagai lingkaran dengan masing-masing mempunyai sudut sebesar 45° , sehingga dalam sebuah lingkaran dibagi menjadi 8 bagian. Perhatikan Gambar 2.11.



Gambar 2.11: Garis lingkaran berada diantara titik (*xk+*1*, yk*)dan (*xk+*1*, yk* − 1)

Anggap bahwa kita mempunyai titik (*xk, yk*). Selanjutnya adalah memilih titik (*xk+*1*, yk*)atau (*xk+*1*, yk* − 1) yang paling mendekati lingkaran. Persamaan untuk lingkaran



Sehingga



Untuk memilih titik (*xk+*1*, yk*)atau (*xk+*1*, yk* − 1) yang paling mendekati lingkaran, kriteria yang digunakan adalah titik tengah (*xk+*1*, yk* − ½ ) yaitu



Jika *p*k< 0 titik tengah berada didalam lingkaran, maka titik (*xk+*1*, yk*)dipilih.

Jika *p*k> 0 titik tengah berada diluar lingkaran, maka titik (*xk+*1*, yk* − 1)dipilih.

Hitung



Atau



Dimana *yk+1* adalah *yk* atau *yk-*1 tergantung tanda dari *pk*. Dalam hal ini kriteria pertama yang dihitung adalah



Jika jari-jari lingkaran bernilai integer (bilangan bulat), kita bisa membulatkan nilai p0 menjadi berikut



Kemudian jika *pk* < 0 maka:



Jika *pk* > 0 maka:



Dari sini diperoleh langkah-langkah algoritma pembentuk lingkaran.

* Langkah-langkah Algoritma Lingkaran Midpoint adalah:

1. Tentukan jari-jari *r* dan pusat lingkaran (*xp*, *yp*), kemudian setting sedemikian rupa sehingga titik awal berada pada: (*x*0, *y*0) = (0 , *r*)
2. Hitung nilai parameter :



Jika jari-jari *r* pecahan



Jika jari-jari *r* bulat

3. Untuk setiap posisi *x*k, dimulai dengan k = 0 berlaku ketentuan:

- bila *p*k *<* 0 maka titik selanjutnya adalah (*x*k+1, *y*k) dan *p*k+1 = *p*k + 2 *x*k+1 + 1

- bila tidak, titik selanjutnya adalah (*x*k+1, *y*k *-* 1) dan *p*k+1 = *p*k + 2 *x*k+1 + 1 – 2 yk+1

1. Tentukan titik simetris pada ketujuh oktan yang lain
2. Gerakkan setiap posisi piksel (*x, y*) pada garis lingkaran dengan titik pusat (xp, yp) dan plot nilai koordinat : *x = x + xp*, *y = y + yp*

6. Ulangi langkah 3 sampai dengan 5 hingga *x* ≥ *y*

Contoh 2.5

Buatlah gambar kurva lingkaran dengan pusat lingkaran (0,0) dan jari-jari 8, perhitungan berdasarkan dari oktan kuadran pertama dimana *x* = 0 sampai *x* = *y*. Koordinat titik awal dimulai dari (*x*,*r*) = (0,8). Karena jari-jari *r* bulat, maka gunakan P0 = 1 – *r*.

Iterasi ke-1:

K = 0 X0 = **0** Y0 = r = **8** P0 = 1 – r = 1 – 8 = –7

Karena P0 < 0, maka :

X1 = X0 + 1 = 0 + 1 = 1 dan Y1 = Y0 = 8, jadi Titik selanjutnya : (1,8)

P1 = *p*0 + 2 *x*1 + 1 = –7 + 2.(1) + 1 = – 4

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(1,8), (–1,8), (1, –8), (–1, –8), (8,1), (–8,1), (8, –1), (–8, –1)

Iterasi ke-2:

K = 1 X1 = 1 Y1 = 8 P1 = – 4

Karena P1 < 0, maka

X2 = X1 + 1 = 1 + 1 = 2 dan Y2 = Y1 = 8, jadi Titik selanjutnya : (2,8)

P2 = *p*1 + 2 *x*2 + 1 = –4 + 2.(2) + 1 = 1

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(2,8), (–2,8), (2, –8), (–2, –8), (8,2), (–8,2), (8, –2), (–8, –2)

Iterasi ke-3:

K = 2 X2 = 2 Y2 = 8 P2 = 1

Karena P2 > 0, maka

X3 = X2 + 1 = 2 + 1 = 3 dan Y3 = Y2 – 1 = 8 –1 = 7 , jadi Titik selanjutnya : (3,7)

P3 = *p*2 + 2 *x*3 + 1 – 2 y3 = 1 + 2.(3) + 1 – 2.(7) = – 6

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(3,7), (–3,7), (3, –7), (–3, –7), (7,3), (–7,3), (7, –3), (–7, –3)

Iterasi ke-4:

K = 3 X3 = 3 Y3 = 7 P3 = – 6

Karena P3 < 0, maka

X4 = X3 + 1 = 3 + 1 = 4 dan Y4 = Y3 = 7, jadi Titik selanjutnya : (4,7)

P4 = *p*3 + 2 *x*4 + 1 = –6 + 2.(4) + 1 = 3

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(4,7), (–4,7), (4, –7), (–4, –7), (7,4), (–7,4), (7, –4), (–7, –4)

Iterasi ke-5:

K = 4 X4 = 4 Y4 = 7 P4 = 3

Karena P4 > 0, maka

X5 = X4 + 1 = 4 + 1 = 5 dan Y5 = Y4 – 1 = 7 –1 = 6 , jadi Titik selanjutnya : (5,6)

P5 = *p*4 + 2 *x*4 + 1 – 2 y4 = 3 + 2.(5) + 1 – 2.(6) = 2

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(5,6), (–5,6), (5, –6), (–5, –6), (6,5), (–6,5), (6, –5), (–6, –5)

Iterasi ke-6:

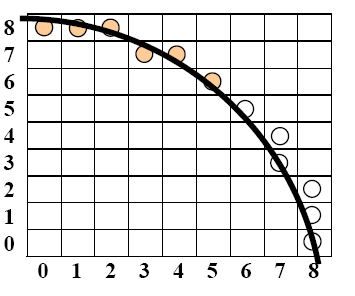
K = 5 X5 = 5 Y5 = 6 P5 = 2

Karena P5 > 0, maka

X6 = X5 + 1 = 5 + 1 = 6 dan Y6 = Y5 – 1 = 6 –1 = 5 , jadi Titik selanjutnya : (6,5)

Iterasi dihentikan karena X > Y.

Bila digambar, hasil untuk oktan ke-1 ditunjukkan oleh gambar (2.12).



Gambar 2.12: Posisi piksel pada pembentukan lingkaran dengan titik pusat (0,0) dan jari-jari 8

Contoh 2.6

Buatlah gambar kurva lingkaran dengan pusat lingkaran P(4,6) dan jari-jari 8.

Jawab:

Berdasarkan contoh 2.5, lingkaran pusat (0, 0) dan jari-jari 8 diperoleh titik-titik pembentuk lingkaran berikut:

pada iterasi ke-1

(1,8), (–1,8), (1, –8), (–1, –8), (8,1), (–8,1), (8, –1), (–8, –1)

Translasikan titik-titik tersebut kearah sumbu x sebesar 4, dan kearah sumbu y sebesar 6, diperoleh titik-titik pembentuk lingkaran dengan pusat P(4, 6) berikut

(5, 14), (3, 14), (5, –2), (3, –2), (12, 7), (–4, 7), (12, 5), (–4, 5)

pada iterasi ke-2

(2,8), (–2,8), (2, –8), (–2, –8), (8,2), (–8,2), (8, –2), (–8, –2)

Translasikan titik-titik tersebut kearah sumbu x sebesar 4, dan kearah sumbu y sebesar 6, diperoleh titik-titik pembentuk lingkaran dengan pusat P(4, 6) berikut

(6, 14), (2, 14), (6, –2), (2, –2), (12, 8), (–4, 8), (12, 4), (–4, 4)

pada iterasi ke-3

(3,7), (–3,7), (3, –7), (–3, –7), (7,3), (–7,3), (7, –3), (–7, –3)

Translasikan titik-titik tersebut kearah sumbu x sebesar 4, dan kearah sumbu y sebesar 6, diperoleh titik-titik pembentuk lingkaran dengan pusat P(4, 6) berikut

(7, 13), (1, 13), (7, –1), (1, –1), (11, 9), (–3, 9), (11, 3), (–3, 3)

pada iterasi ke-4

(4,7), (–4,7), (4, –7), (–4, –7), (7,4), (–7,4), (7, –4), (–7, –4)

Translasikan titik-titik tersebut kearah sumbu x sebesar 4, dan kearah sumbu y sebesar 6, diperoleh titik-titik pembentuk lingkaran dengan pusat P(4, 6) berikut

(8, 13), (0, 13), (8, –1), (0, –1), (11, 10), (–3, 10), (11, 2), (–3, 2)

pada iterasi ke-5

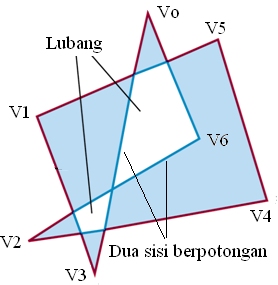
(5,6), (–5,6), (5, –6), (–5, –6), (6,5), (–6,5), (6, –5), (–6, –5)

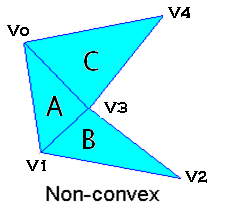
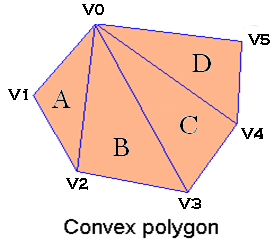
Translasikan titik-titik tersebut kearah sumbu x sebesar 4, dan kearah sumbu y sebesar 6, diperoleh titik-titik pembentuk lingkaran dengan pusat P(4, 6) berikut

(9, 12), (–1, 12), (9, 0), (–1, 0), (10, 11), (–2, 11), (10, 1), (–2, 1)

2.5 Polygon

Polygon adalah kumpulan garis lurus yang saling menyambung hingga membentuk suatu luasan. Garis-garis ini disebut *edge* (sisi polygon). Titik pertemuan tiap dua sisi disebut verteks. Biasanya polygon dinyatakan dengan koordinat verteks-verteks ini. Ada dua jenis polygon yaitu polygon sederhana dan polygon tidak sederhana. Ciri polygon sederhana adalah tidak semua verteks berada pada bidang yang sama, tidak mempunyai sisi-sisi yang berpotongan, dan tidak mempunyai lubang.



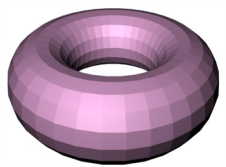


Gambar 2.13 : Polygon sederhana dan polygon tidak sederhana

1. Polygon Tidak sederhana

(b) Polygon sederhana

Polygon sederhana dibagi menjadi dua yaitu convex polygon dan non convex polygon. Ciri convex polygon adalah semua sudut interiornya < 180o, atau setiap segmen garis yang dihasilkan dari dua buah verteks sembarang dalam polygon berada didalam polygon. Bentuk polygon yang paling sederhana adalah segitiga, karena semua polygon dapat dipecah-pecah menjadi bagian yang terkecil yaitu segitiga seeprti pada Gambar 2.13(b). Mengapa polygon ? Dengan polygon secara praktek kita bisa melakukan pendekatan untuk membentuk permukaan setiap obyek 3-D, jika kita mempunyai jumlah polygon yang cukup. Sebagai contoh permukaan bola, torus, dan teko seperti pada Gambar 2.14 dapat dibuat dari beberapa polygon.



Gambar 2.14: Permukaan bola, torus dan teko yang terbentuk dari banyak polygon.

2.6 *Filling Polygon*

Dalam grafis, pemberian warna dibutuhkan untuk mempercantik tampilan polygon. Karena itu diperlukan algoritma khusus untuk mengisi warna pada polygon tersebut. Teknik atau algoritma untuk pengisian warna pada polygondisebut *filling polygon* atau *area filling*. Ada dua macam dasar pendekatan *filling* *polygon* pada sistem raster yaitu scan-line *Polygon Fill Algorithm* dan *Boundary-Fill Algorithm*:

2.6.1 *Scan Line Polygon Fill Algorithms*

Pemberian warna pada polygon dilakukan dengan cara men-*scan* secara horisontal dari kiri ke kanan. Hal ini dilakukan untuk mendapatkan titik potong dengan tepi polygon, kemudian mengurutkan nilai-nilai titik potong *x* dari kiri ke kanan dan memberi warna pada piksel-piksel diantara dua pasangan berurutan (*x*1- *x*2). Hal ini dilakukan dari garis scan yang paling bawah (nilai y terkecil)hingga garis scan yang paling atas seperti pada Gambar 2.15. Metode ini bisa juga digunakan untuk pengisian warna pada obyek-obyek sederhana lainnya, misalnya lingkaran, ellip dan lain-lain.

Scanline terakhir

Piksel-piksel diantara titik potong *x*1-*x*2

Scanline pertama

Titik potong *x*2

Titik potong *x*1

Gambar 2.15: Metode scan-line

Bagaimana kita tahu bahwa piksel tersebut berada didalam polygon atau diluar polygon ?

* *Inside-Outside test*

Perhatikan Gambar 2.16. Untuk mengidentifikasi bagian dalam dan bagian luar digunakan aturan paritas ganjil-genap yang biasa disebut sebagai *odd-even rule* atau *odd parity rule*. Semula kita men-set paritas dengan nilai genap. Setiap ditemukan titik potong nilai paritas dibalik, yang semula genap dibalik menjadi ganjil, sebaliknya yang semula ganjil dibalik menjadi genap. Beri warna pada piksel jika paritasnya Ganjil.

paritas ganjil

paritas genap

Gambar 2.16: aturan paritas ganjil-genap untuk mengisi warna

**Masalah:** Pada Gambar 2.17(a)**,** verteks **a, b, c,** dan **d** merupakan pertemuan dari dua buah segmen garis. Mengapa pada verteks **a** dan **d** dihitung sekali, sedangkan pada verteks **b** dan **c** dihitung dua kali ?

2

1

2

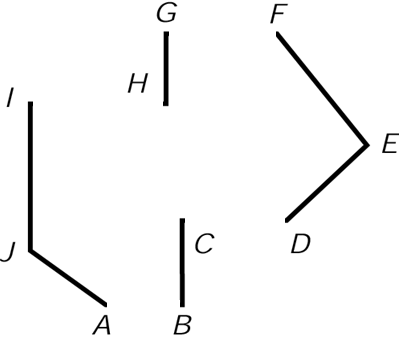
1

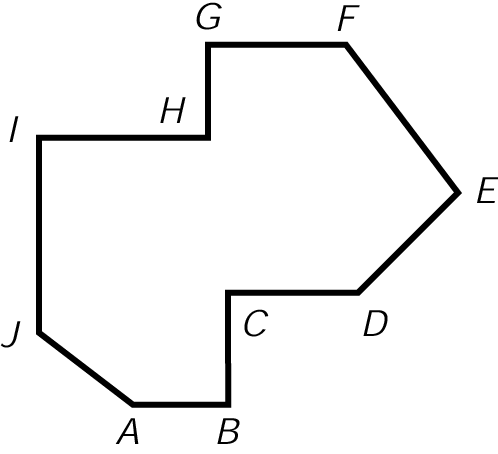
1. (b)

Gambar 2.17: perhitungan paritas pada verteks **a, b, c,** dan **d.**

**Solusi:** Buatlah perputaran tepi polygon searah jarum jam seperti pada Gambar 2.17(b). Check, apabila pada suatu verteks arah anak panahnya berubah (pada verteks **b** dan **c** yaitu: naik-turun atau turun-naik) maka pada verteks tersebut dihitung dua kali. Sebaliknya jika arah anak panahnya tidak berubah (pada verteks **a** dan **d** yaitu: naik-naik atau turun-turun) maka pada verteks tersebut dihitung sekali.

**Masalah:** Perhatikan Gambar 2.18(a),Bagaimana perhitungan tepi horisontal **AB, CD, GF,** dan **HI** ?





1. (b)

Gambar 2.18: polygon dengan sisi-sisi horisontal

**Solusi:** abaikan verteks-verteks yang terletak pada tepi horisontal, atau jangan dimasukkan dalam perhitungan paritas ganjil-genap, sehingga tampak seperti pada Gambar 2.16(b).

Algoritma scanline menggunakan kaidah paritas ganjil-genap:

* Tentukan titik potong garis scan dengan semua sisi polygon
* Urutkan titik potong tadi berdasarkan koordinat sumbu *x*
* Warnai semua piksel diantara pasangan titik potong (*x*1-*x*2) yang terletak didalam polygon menggunakan kaidah paritas ganjil-genap.

for each scanline

edgeCnt = 0;

for each pixel on scanline (l to r)

if (oldpixel->newpixel crosses edge)

edgeCnt ++;

// draw the pixel if edgeCnt odd

if (edgeCnt % 2)

setPixel(pixel);

Kelemahan algoritma ini adalah : memerlukan biaya tinggi (*big cost*) karena pada setiap sisi polygon selalu dilakukan pengujian terhadap piksel-piksel.

**Solusi:** menggunakan konsep *edge table* (ET)

* ***Edge Table* ( ET )**

*edge table* (ET) adalah tabel yang berisi …………

Kita akan menggunakan dua *edge table* yang berbeda, yaitu:

*Active Edge Table* (AET) : menyimpan semua sisi polygon yang berpotongan dengan garis scan.

*Global Edge Table* (GET) : menyimpan semua sisi polygon dan digunakan untuk meng-*update* AET.

**Perhitungan titik potong dengan garis scan**

Kita tahu bahwa:



Setiap garis scan yang baru berlaku :



Kita perlu menghitung *x* untuk setiap garis scan:



Sehingga,



Dan



Maka



Ini adalah cara yang efisien untuk menghitung nilai *x.*

***Active Edge Table* (AET)**

* Tabel berisi satu entry per sisi yang berpotongan dengan garis scan aktif.
* Pada setiap garis scan yang baru :



* + Hitung titik potong baru untuk semua sisi menggunakan rumus :
  + Tambahkan setiap sisi baru yang berpotongan
  + Hapus setiap sisi yang tidak berpotongan
* Untuk efisiensi Update AET, kita tetap menjaga GET.

***Global Edge Table* (GET)**

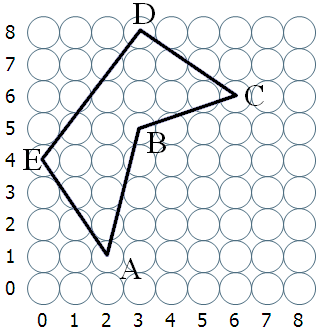
* Tabel yang berisi informasi tentang semua sisi-sisi polygon
* GET mempunyai satu tempat unttuk setiap garis scan
  + Setiap tempat menyimpan daftar sisi yang mempunyai nilai *y*min
  + Setiap sisi ditentukan hanya dalam satu tempat
* Tiap-tiap entry dalam GET berisi
  + Nilai *y*max dari sisi
  + Nilai *x*@ymin (nilai *x* pada titik *y*min)
  + Nilai pertambahan *x* (1/m)

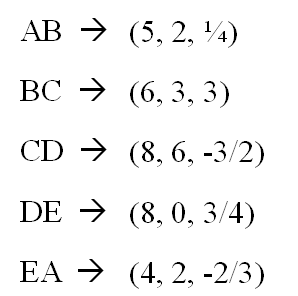
***Scan Line Polygon Fill Algorithms* menggunakan ET, GET dan AET**

1. Tambahkan sisi-sisi polygon ke GET
2. Set *y* ke koordinat *y* terkecil dalam GET
3. Inisialisasi, set AET = kosong
4. Ulang sampai AET dan GET kosong
   1. Tambahkan sisi-sisi dari GET ke AET yang mana *y*min = *y*.
   2. Hilangkan sisi-sisi dari AET bila *y*max = *y*.
   3. Urutkan AET berdasarkan *x*
   4. Warnai piksel yang terletak diantara pasangan titik potong dalam AET
   5. Untuk setiap sisi dalam AET, ganti *x* dengan *x* + 1/m
   6. Set *y* = *y* + 1 untuk bergerak ke garis scan berikutnya

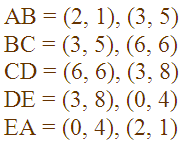
Contoh 2.7

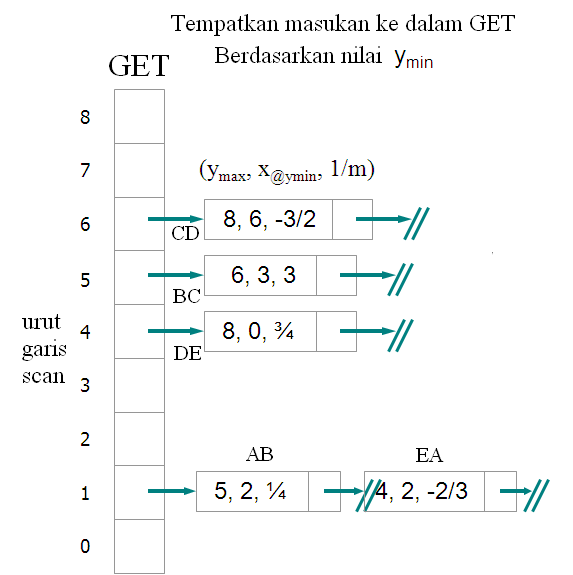
Diketahui polygon berikut, gunakan konsep *edge table* untuk mewarnai polygon tersebut.

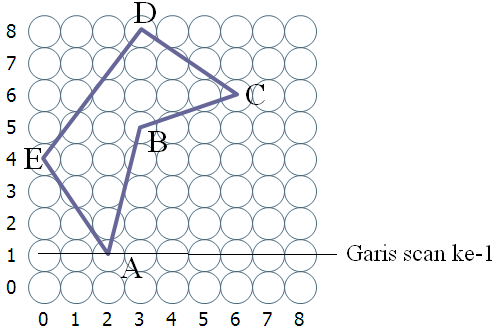




Sisi-sisi pembentuk polygon







*y*max, *x*,1/m

AET

EA

5, 2, 1/4

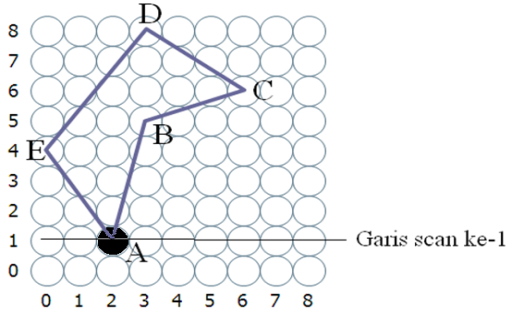
AB

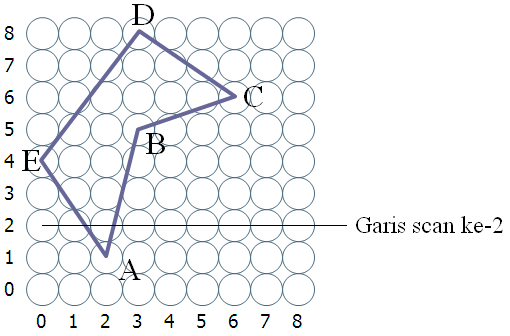
4, 2, −2/3

*x*kanan = 2

*x*kiri = 2

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (2 − 2), hasilnya adalah





xi+1 = xi + 1/m

= 2 – 2/3

= 4/3

xi+1 = xi + 1/m

= 2 + 1/4

= 9/4

*y*max, *x*,1/m

AET

EA

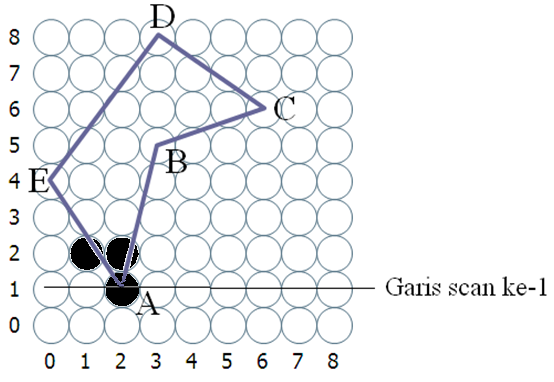
AB

5, 9/4, 1/4

4, 4/3 , −2/3

*x*kiri  = 4/3 ≈ 1 dan *x*kanan = 9/4 ≈ 2

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (1 − 2), hasilnya adalah



xi+1 = xi + 1/m

= 4/3 – 2/3

= 2/3

xi+1 = xi + 1/m

= 9/4 + 1/4

= 10/4

*y*max, *x*,1/m

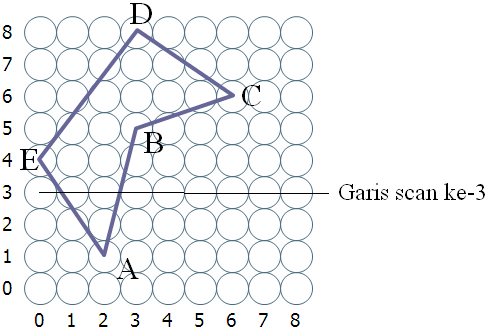
AET

EA

AB

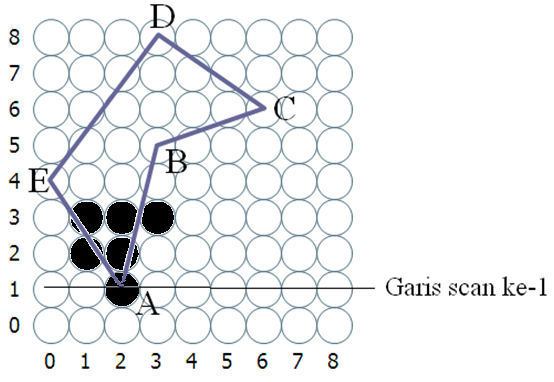
5, 10/4, 1/4

4, 2/3 , −2/3



*x*kiri  = 2/3 ≈ 1 dan *x*kanan = 10/4 ≈ 3

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (1 − 3), hasilnya adalah



pada EA, *y*max = 4. EA harus dihapus dari AET. Dalam GET sisi DE tersimpan pada *y* = 4. Jadi sisi EA diganti dengan sisi DE

xi+1 = xi + 1/m

= 10/4 + 1/4

= 11/4

*y*max, *x*,1/m

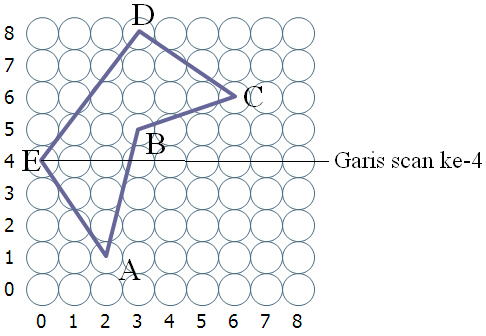
AET

~~EA~~ = DE

AB

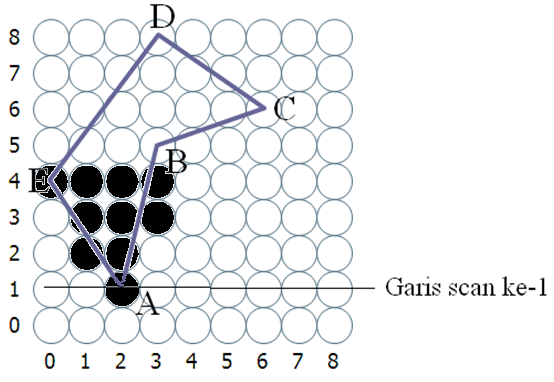
5, 11/4, 1/4

8, 0 , 3/4



*x*kiri  = 0 dan *x*kanan = 11/4 ≈ 3

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (0 − 3), hasilnya adalah



pada AB, *y*max = 5. AB harus dihapus dari AET. Dalam GET sisi BC tersimpan pada *y* = 5. Jadi sisi AB diganti dengan sisi BC

xi+1 = xi + 1/m

= 0 + 3/4

= 3/4

*y*max, *x*,1/m

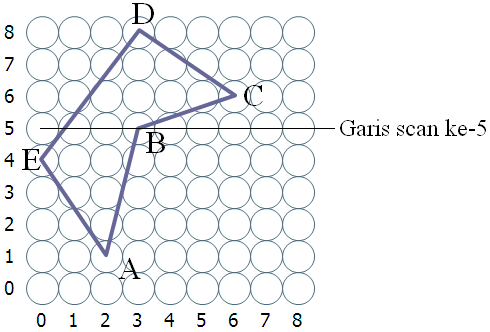
AET

DE

~~AB~~ = BC

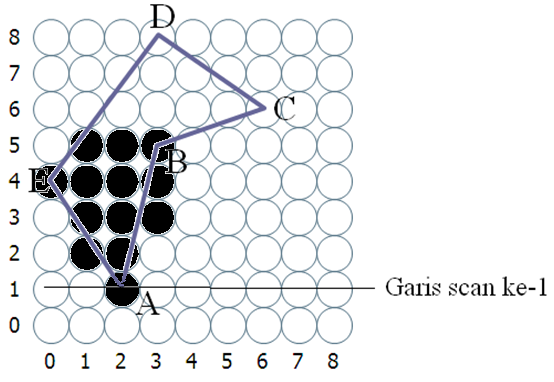
6, 3, 3

8, 3/4 , 3/4



*x*kiri  = 3/4 ≈ 1 dan *x*kanan = 3

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (1 − 3), hasilnya adalah



pada BC, *y*max = 6. BC harus dihapus dari AET. Dalam GET sisi CD tersimpan pada *y* = 6. Jadi sisi BC diganti dengan sisi CD

xi+1 = xi + 1/m

= 3/4 + 3/4

= 6/4

*y*max, *x*,1/m

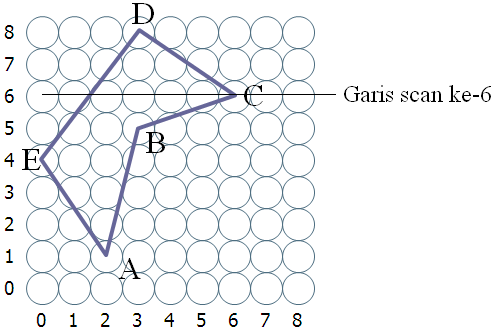
AET

DE

~~BC~~ = CD

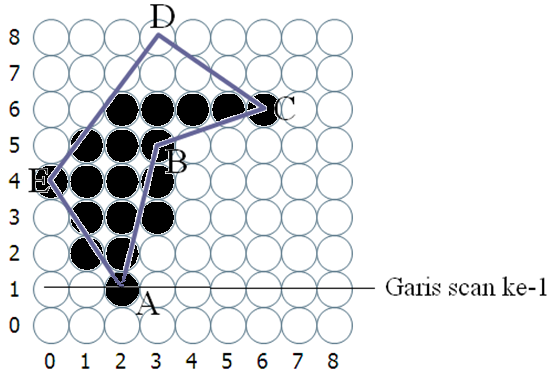
8, 6, −3/2

8, 6/4 , 3/4



*x*kiri  = 6/4 ≈ 2 dan *x*kanan = 6

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (2 − 6), hasilnya adalah



xi+1 = xi + 1/m

= 6 − 3/2

= 9/2

xi+1 = xi + 1/m

= 6/4 + 3/4

= 9/4

*y*max, *x*,1/m

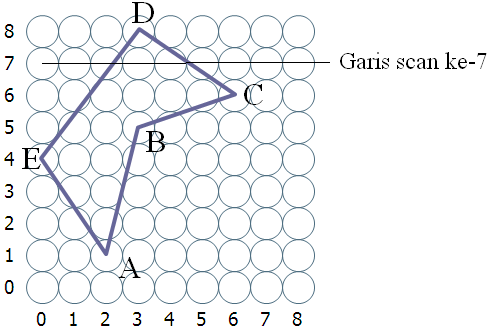
AET

DE

CD

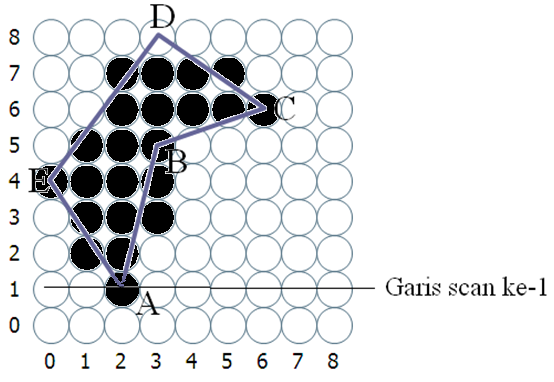
8, 9/2, −3/2

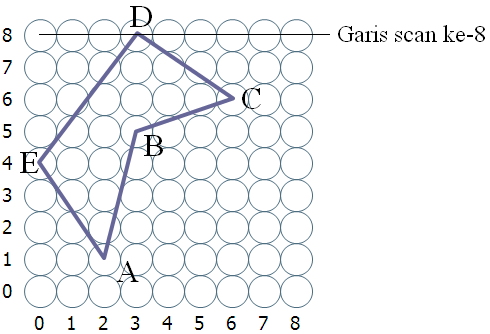
8, 9/4 , 3/4



*x*kiri  = 9/4 ≈ 2 dan *x*kanan = 9/2 ≈ 5

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (2 − 5), hasilnya adalah





xi+1 = xi + 1/m

= 9/2 − 3/2

= 3

xi+1 = xi + 1/m

= 9/4 + 3/4

= 3

*y*max, *x*,1/m

AET

DE

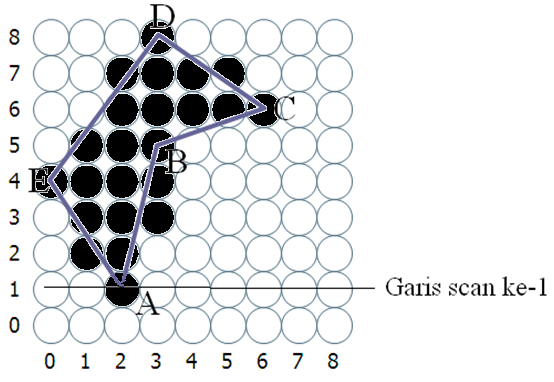
CD

8, 3, −3/2

8, 3, 3/4

*x*kiri  = 3 dan *x*kanan = 3

Pewarnaan dilakukan diantara titik potong (*x*kiri − *x*kanan) = (3 − 3), hasilnya adalah



Karena sisi polygon dalam GET ataupun AET sudah habis, maka proses dihentikan.

* + 1. ***Boundary-Fill Algorithm***

Prosedur *boundary-fill* menerima tiga parameter yaitu: koordinat titik (*x,y*), warna isi dan warna garis batas. Proses pengisian warna tertentu dimulai dari titik (*x,y*), kemudian memeriksa posisi titik tetangganya, apakah titik tetangga tersebut memiliki warna batas:

* + Jika tidak, warnai titik tersebut dengan warna tertentu.
  + Selanjutnya periksa lagi posisi dan warna titik tetangganya.
  + Proses diulangi terus hingga seluruh titik pada area pengisian telah diuji.

Dengan teknik ini pengisian warna dimulai dari sebuah titik yang berada didalam area (*x,*y) polygon dan mewarnai piksel dari titik tetangganya hingga semua piksel yang berada didalam polygon telah diwarnai seperti pada Gambar 2.19.

Gambar 2.19: Metode *Boundary-Fill*

Ada 2 macam cara untuk melihat titik tetangga, yaitu :

* + 4 tetangga, melihat titik yang berada diatas, bawah, kanan dan kiri
  + 8 tetangga, melihat titik yang berada diatas, bawah, kanan, kiri, pojok kiri atas, pojok kiri bawah, pojok kanan atas dan pojok kanan bawah.

a) 4- tetangga

1

4

X

2

3

b) 8-tetangga

1

2

3

8

X

4

7

6

5

Contoh 2.8

diketahui : polygon = {(1,1), (2,,5), (5,4), (8,7), (10,4), (10,2), (1,1)}, lakukan *Area Filling* menggunakan algoritma *Boundary Fill Algorithm* 4-tetangga.

Jawab:

titik-titik sebagai pembentuk polygon = {(1,1), (2,,5), (5,4), (8,7), (10,4), (10,2), (1,1)}.

Bila poligon tersebut digambar, diperoleh gambar berikut :

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

22

1

Misalkan titik awal pencarian adalah (3,3). Tandai titik (3,3) dengan warna tertentu, misalnya warna merah. Lihat 4-tetangganya, yaitu titik (3,2), (3,4), (2,3), (4,3).

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

22

1

Ke-4 tetangga tersebut bukan garis batas poligon, sehingga 4-titik tersebut diwarnai merah.

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

22

1

Titik yang telah diproses: (3,3)

Titik yang belum diproses : (3,2), (3,4), (2,3), (4,3)

Ambil titik (3,2).

Titik yang telah diproses: (3,2), (3,3)

Titik yang belum diproses : (3,4), (2,3), (4,3)

4-tetangga titik tersebut adalah (3,3), (3,1), (2,2), (4,2). Terlihat bahwa titik (4,2) dan (2,2) bukan garis batas poligon, sehingga diwarnai dengan warna merah. Titik (3,3) sudah diwarnai. Titik (3,1) adalah garis batas jadi tidak diwarnai.

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

3

22

1

Titik yang telah diproses: (3,3)(3,2)

Titik yang belum diproses : (3,4), (2,3), (4,3) (2,2), (4,2)

Ambil titik (3,4). 4-tetangga titik tersebut adalah (3,3), (3,5), (2,4), (4,4). Titik (3,3) sudah diwarnai. Titik (3,5), (2,4) dan (4,4) adalah garis batas jadi tidak diwarnai.

Proses diulang sehingga seluruh bagian dalam poligon diwarnai dengan warna merah.

1 2 3 4 5 6 7 8 9 10 11

7

6

5

4

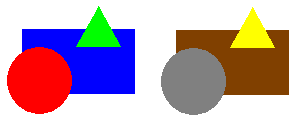
3

22

1

2.6.3 *Flood-Fill Algorithm*

Terkadang kita ingin mewarnai ( atau memberi warna yang baru) pada sebuah area yang mempunyai warna lebih dari satu. Perhatikan gambar 2.12 berikut



(a)

(b)

Gambar 2.12: penggantian warna obyek menggunakan *Flood-Fill Algorithm* . (a) lingkaran berwarna merah, segitiga berwarna hijau dan persegi berwarna biru. Obyek tersebut warnanya diubah menjadi (b) lingkaran berwarna abu-abu, segitiga berwarna kuning dan persegi berwarna coklat

Algoritma ini dimulai dari titik yang berada didalam area (*x,y*) dan mengganti semua pikselnya dengan warna baru sehingga bagian dalam area mempunyai warna yang sama. Pengujian titik tetangga bisa menggunakan 4-tetangga atau 8-tetangga.

Ringkasan

Output primitif adalah struktur dasar geometri yang paling sederhana dari gambar grafika komputer. Titik dan garis adalah contoh dari output primitif yang dapat digunakan untuk membentuk gambar, misalnya lingkaran, kerucut, permukaan berbentuk persegi, kurva dan permukaan berbentuk lengkung, warna area dan karakter, dan lain-lain.

*Piksel* adalah elemen gambar terkecil berupa sebuah titik yang ditempatkan dilayar. warna merupakan atribut dari piksel.

***Frame buffer* adalah**  area memory tempat informasi gambar disimpan.

Ada 3 algoritma untuk membentuk garis, yaitu Algoritma *brute force*, DDA, dan Bressenham. Sedangkan untuk membentuk lingkaran digunakan algoritma lingkaran midpoint.

Polygon adalah kumpulan garis lurus yang saling menyambung hingga membentuk suatu luasan. Garis-garis ini disebut *edge* (sisi polygon). Titik pertemuan tiap dua sisi disebut verteks.

Untuk mengisi warna pada polygon diperlukan algoritma khusus. Teknik atau algoritma untuk pengisian warna pada polygondisebut *filling polygon* atau *area filling*. Ada dua macam dasar pendekatan *filling* *polygon* pada sistem raster yaitu scan-line *Polygon Fill Algorithm* dan *Boundary-Fill Algorithm*:

Soal-Soal Latihan

1. Apa yang dimaksud dengan output primitif ?

2. Bagaimana cara membuat gambar garis dan gambar lingkaran menggunakan output primitif titik ?

3. Diketahui 2 buah titik A(5,3) dan titik B(15,8) bila titik A sebagai titik awal dan titik B sebagai titik akhir, tentukan titik-titik antara yang menghubungkan titik A dan titik B sehingga membentuk garis AB dengan menggunakan (a) Algoritma *brute force* (b) algoritma DDA (c) algoritma Bressenham (d) Bagaimana menurut saudara, mana yang lebih baik, algoritma DDA atau algoritma Bressenham ? Mengapa ?

4. (a) Buatlah gambar kurva lingkaran dengan pusat lingkaran (0,0) dan jari-jari 6, perhitungan berdasarkan dari oktan kuadran pertama dimana *x* = 0 sampai *y* = *r*. Koordinat titik awal dimulai dari (*x*,*r*) = (0,6). Untuk mempermudah perhitungan gunakan P0 = 1 – r ( sekali lagi, ini hanya untuk mempermudah perhitungan dalam contoh). (b) sama seperti soal (a), tetapi pusat lingkarang di P(2,5).

5. Diketahui : polygon = {(2,1), (3,6), (5,4), (8,8), (10,4), (12,2), (2,1)}, lakukan *Area Filling* menggunakan (a) algoritma *Scan Line Polygon* (b) algoritma *Boundary Fill*.