INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
School of Telematics

IF-311
Grafik Komputer

Yohanes Nugroho, ST

Daftar Isi

BAB I
PERKENALAN GRAFIKA KOMPUTER

Grafika Komputer merupakan bidang yang berhubungan dengan penanganan grafik vektor pada komputer. Penanganan grafik yang berhubungan dengan bitmap masuk dalam bidang pengolahan citra. Meskipun ada perbedaan dalam kedua bidang tersebut, namun terkadang keduanya dianggap dalam bidang yang sama yaitu grafika komputer.

Grafik vektor adalah grafik yang dibuat berdasarkan informasi geometri suatu objek sedangkan grafik bitmap adalah grafik yang dibuat berdasarkan informasi titik-titik yang disusun berdasarkan hasil digitasi. Penanganan grafik vektor meliputi transformasi vektor sedangkan penanganan bitmap meliputi operasi terhadap titik gambar (misalnya enhance, blur, dll).

Output Grafik

Output Grafik bisa dibagi menjadi dua kategori, yang permanen dan yang non permanen. Dulu output grafik hanya terbatas pada pembuatan garis baik melalui sinar untuk monitor ataupun melalui peralatan plotter untuk output permanen. Namun dalam perspektif teknologi saat ini semua bisa dianggap sama yaitu sekedar meletakkan titik pada koordinat tertentu.

Alat-alat yang ada saat ini umumnya adalah device raster scan. Dalam device raster scan gambar dibuat baris per baris dari atas ke bawah dengan terdiri atas titik-titik diskrit perbaris, dengan setiap titik dapat diset warna dan intensitasnya secara individual.

Karena penggunaan teknologi raster, maka ada keperluan untuk mengubah gambar yang sifatnya kontinu (misalnya garis) menjadi sejumlah titik untuk merepresentasikan gambar tersebut. Proses ini disebut dengan scan conversion.

Primitif Grafik dan Atribut

Gambar terumit yang dihasilkan oleh grafika komputer dihasilkan oleh primitif grafik yang relatif kecil. Primitif grafik yang biasanya ada dalam suatu paket grafik adalah:

1. sebuah titik
2. garis dengan suatu titik awal dan akhir
3. polyline, atau serangkaian garis
4. polygon berisi
5. teks

Mungkin ada primitif lain yang disediakan, tapi tidak esensial seperti persegi panjang, lingkaran, kurva, dll.

Setiap primitif grafik memiliki suatu set attribut. Misalnya titik memiliki warna, garis memiliki warna, style (putus-putus, terisi penuh).

Menghasilkan Gambar

Untuk menghasilkan suatu gambar, ada tiga proses dasar yang perlu dilakukan

1 Permodelan

Sebelum menghasilkan gambar suatu objek, kita perlu memiliki definisi property objek, misalnya: list permukaan, list simpul (node). Dan property fisiknya, seperti warnanya. Informasi total mengenai suatu objek disebut sebagai model.

Model dalam grafika komputer bisa terdiri dari sekedar array dua dimensi yang menyatakan ujung-ujung garis, sampai dengan model kompleks yang terdiri dari banyak polygon dan informasi pencahayaan untuk masing-masing polygon seperti pada program simulasi 3D.

Proses untuk menciptakan deskripsi objek disebut dengan permodelan.

Viewing

Sebelum gambar dari suatu model dapat ditampilkan, kita harus mendefinsikan view dari model. Ini biasanya mengindikasikan di mana kamera virtual harus diletakkan dan ke arah mana kamera itu menghadap. Informasi mengenai lebarnya jarak pandang juga harus didefinisikan (misalnya apakah wide screen atau telefoto).

Rendering

Setelah model didefinisikan dan informasi view juga sudah didefinisikan kita bisa menghasilkan gambar final. Proses menghasilkan gambr ini disebut dengan rendering. Proses-proses yang termasuk dalam rendering adalah: menghilangkan permukaan yang tidak terlihat, menhitung bayangan permukaan yang terlihat, dan melakukan scan converting untuk menampilkan hasilnya.

Paket Grafik

Fasilitas grafik umumnya diimplementasikan sebagai paket fungsi yang bisa dipakai di program. Setiap paket grafik memiliki level kerumitannya masing-masing. Dalam level yang terendah sebuah garis bisa digambar dengan mengeset lokasi memori individual yang merepresentasikan garis pada device raster scan. Kebanyakan programmer tidak lagi harus berurusan dengan hal-hal seperti ini, bahkan sebagian proses tersebut sudah dilakukan di level hardware. Namun dalam kuliah ini akan tetap dibahas aneka algoritma grafik dasar untuk lebih memahami proses yang terjadi, dan pemahaman algoritma dasar ini bisa dipakai pada hal lain.

Beberapa paket grafik yang tersedia saat ini adalah:Xlib, DirectX, GKS, XG, PHIGS, OpenGL. Masing-masing paket grafik memiliki kelebihan dan kekurangan masing-masing, baik ditinjau dari kelengkapan paketnya, kecepatannya, maupun dukungan sistem operasi yang didukungnya.

BAB II
HARDWARE UNTUK GRAFIKA KOMPUTER

Algoritma-algoritma yang dipelajari dalam mata kuliah grafika komputer mengasumsikan hardware yang "sempurna", yang pada kenyataannya tidak ditemukan dalam kehidupan sehari-hari. Batasan-batasan pada device membuat kita tidak bisa menghasilkan garis miring yang lurus sempurna, atau gambar foto yang seindah aslinya.

Dalam bab ini akan dibahas berbagai jenis device yang berhubungan dengan input, output, dan pemrosesan grafik. Hal-hal yang akan dibahas meliputi cara kerja device, karakteristik, dan batasan yang dimiliki oleh sebuah device.

Hardware output

Dalam kuliah grafika komputer, device output biasanya merupakan yang paling penting. Fokus dalam kuliah grafika adalah pada algoritma, dengan hasil yang bisa dilihat pada output, input untuk menguji suatu algoritma biasanya tidak perlu kompleks atau interaktif .

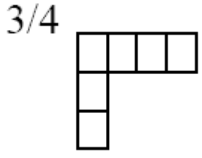
Hal yang terpenting dalam output adalah bahwa output grafik sifatnya terbatas, terutama dalam hal resolusi output. Algoritma-algoritma grafik yang akan dipelajari umumnya akan menggunakan keterbatasan device output ini sebagai batasan dalam implementasi algoritma.

Istilah istilah dalam output grafik

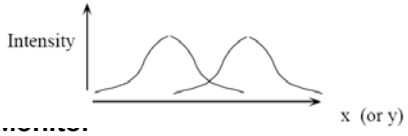
Ada beberapa istilah dalam output grafik yang perlu diketahui, beberapa istilah hanya berlaku untuk device tertentu (misal: refresh rate untuk monitor CRT), tapi beberapa berlaku umum. Istilah-istilah yang umum meliputi:

Pixel
Satu elemen titik gambar, baik itu di layar, di printer, maupun di media lain

Aspect Ratio
Rasio jumlah piksel vertikel dengan horizontal untuk menghasilkan garis dengan panjang yang sama



ksimum yang dapat ditampilkan di device tanpa overlap x 768, dll), atau Jumlah titik yang bisa ditampilkan per inch atau per milimeter (300 dpi, 1200 dpi). Overlap didefinisikan sebagai bagian piksel yang mengambil tempat yang sama kurang dari 60%



Monitor

Monitor merupakan alat output yang paling umum digunakan. Saat ini ada beberapa jenis monitor, CRT, LCD dan Plasma. Selama 75 tahun dan sampai hari ini, teknologi CRT masih yang paling banyak dipakai.

CRT

Pada teknologi CRT, gambar dilukis dilayar dan akan segera hilang, sehingga perlu terus menerus direfresh. Dalam dunia CRT dikenal istilah refresh rate, refresh rate merupakan ukuran seberapa sering perlu merefresh setiap titik di layar.

Refresh rate 60 Mhz merupakan nilai yang umum yang artinya setiap titik direfresh sekali setiap 16667 mikrosekon dan persistensi untuk sebuah titik biasanya 10-60 mikrosekon (artinya setelah dilukis, lukisan hanya akan bertahan selama 10-60 mikrosedon). Jadi sebenarnya setiap titip lebih sering "gelap" dibanding terang, namun mata kita tidak bisa melihat itu. Kita bisa melihat ini dengan jelas jika kita melihat monitor (CRT biasa, dengan refresh rate tertentu) di layar televisi. Apabila keseluruhan monitor tampak berkedip (flicker) biasanya hal tersebut disebabkan oleh refresh rate yang kurang tinggi.

LCD

LCD (liquid cystal display) bekerja berdasarkan kristal yang berubah polaritasnya jika dialiri listrik. Beberda dengan CRT, polaritas kristal tidak berubah ketika masih dialiri listrik, sehingga LCD tidak memerlukan refresh (tidak ada refresh rate dalam LCD). Jika

Printer dan Plotter

Selain teknologi untuk menampilkan gambar pada suatu saat, kita juga perlu mengetahui teknologi untuk menghasilkan gambar permanen dalam bentuk hardcopy. Kecuali daisy wheel (yang sudah tidak ada lagi), printer mencetak per titik, dan hal ini berarti bahwa citra yang ditampilkan juga kadang tidak sempurna.

Impact printer

Impact printer adalah printer yang menggunakan cara mekanik untuk menuliskan citra pada media yang dicetak. Ada dua jenis impact printer yaitu dot matrix dan daisy wheel. Daisy wheel merupakan teknologi kuno, dimana head printer berbentuk seperti huruf-huruf dalam mesin ketik, meskipun dapat menghasilkan karakter dengan cukup cepat dan dengan kualitas yang baik, namun daisy wheel tidak praktis karena tidak cukup generik (tidak bisa mencetak grafik).

Printer dot matrix

Sesuai dengan namanya, head printer adalah matrix dari pin dan pin ini bisa ditarik untuk membuat/mencetak pola tertentu. Akurasi printer dot matrix adalah yang paling rendah dibanding teknologi printer yang lain, dan tidak bisa digunakan untuk membuat grafik kualitas tinggi. Namun printer ini paling cepat untuk teks (karena pola font yang sudah dihardcode) dan dapat digunakan untuk membuat banyak salinan dengan kertas karbon (yang paling ekonomis)

Non Impact printer

Non Impact printer adalah printer yang menggunakan cara non mekanis (laser, semprotan tinta) untuk menghasilkan citra pada media tercetak. Ada banyak jenis non impact printer: Laser, ink jet, xerografik, electro static, electro thermal. Dua jenis yang paling populer adalah laser dan ink jet.

Laser Printer

Laser Printer adalah printer jenis non impact yang menggunakan laser untuk

memberi muatan pada kertas yang akan menarik serbuk tinta. Printer Laser memberikan hasil terbaik namun dengan harga yang paling mahal.

Inkjet dan Bubble Jet

Inkjet dan Bubble jet memiliki prinsip kerja yang sama: tinta disemprotkan dalam kecepatan tinggi dalam butiran-butiran kecil. Inkjet merupakan printer tingkat menengah yang bisa menyediakan kualitas cukup tinggi (walaupun tidak setinggi laser), namun tetap tidak bias menggantikan printer impact (karena tidak bisa membuat banyak salinan sekaligus).

Hardware Input

Untuk memasukkan objek grafik ke dalam komputer, maka diperlukan alat input tertentu. Alat input ini bisa memasukkan objek gambar secara visual-interaktif ataupun manual dengan memasukkan koordinat dan informasi lain gambar (warna,, tekstur, dll).

Keyboard

Keyboard merupakan alat yang paling konvensional untuk input grafik. Alat ini ada di hampir semua PC. Keyboard cocok digunakan untuk input koordinat (misalnya pada design 3D), ataupun input lain yang berupa perintah (misalnya "insert rectangle...").

Mouse

Mouse merupakan alat penunjuk yang juga bisa memeberikan input tertentu (tergantung alatnya, biasanya berupa tombol namun ada juga yang memberikan input numerik). Ada dua jenis mouse saat ini, yang mekanik (menggunakan bola) dan optik (menggunakan laser). Mouse bukan merupakan cara input terbaik untuk grafik (presisi cukup rendah), namun ini merupakan yang paling banyak dipakai.

Digitizer Tablet

Digitizer tablet merupakan alat input berupa pena dan permukaan khusus. Dengan menggunakan digitizer tabel, seorang seniman bisa "melukis" atau "menggambar" di atas digitizer tablet seperti menggambar di atas kertas.

Scanner dan Kamera Digital

Alat-alat yang disebutkan sebelumnya digunakan untuk memasukkan gambar baru. Scanner dan kamera digital digunakan untuk memasukkan gambar yang sudah ada. Scanner berfokus pada kertas dan media tercetak lain, sedangkan kamera digital digunakan untuk menangkap gambar panorama.

Prosessor Grafik

Prosessor grafik merupakan hardware pembantu untuk memperoleh kecepatan yang lebih tinggi dalam memproses grafik. Secara teori, apa yang dilakukan oleh prosessor grafik bisa dilakukan oleh prosessor, namun prosessor akan menjadi lambat jika hanya menangani komputasi grafik tersebut.

Bagaimana prosessor grafik bekerja? Pada level tertentu di sistem operasi (atau di level user), ada operasi-operasi dasar yang disediakan oleh library tertentu (DirectX, Open-GL, SDL, dll), dan programmer hanya perlu mengunakan fungsi ini. Tanpa prosessor grafik, operasi-operasi tersebut dikerjakan langsung oleh

prosessor, sedangkan jika kita menginstall prosessor grafik, library (atau bagian driver dari library) tersebut akan menggantikan fungsi perhitungan dengan memberikan instruksi-instruksi khusus ke prosessor grafik.

BAB III
JAVA UNTUK GRAFIKA KOMPUTER

Untuk mengimplementasikan algoritma-algoritma yang dipelajari dalam mata kuliah grafika komputer kita perlu memilih suatu lingkungan pengembangan. Lingkungan ini harus bisa menampilkan grafik dan memungkinkan interaksi dengan alat input grafik dengan mudah.

Keperluan dalam belajar pemrograman Grafik

Ada beberapa hal yang diperlukan dalam belajar pemrograman grafik, yang meskipun tidak mutlak, dapat membantu lebih mengerti algoritma grafik

1. Akses atau abstraksi ke piksel dalam layar
2. Cross platform untuk melihat efek grafik dalam berbagai lingkungan (misal: efek tidak adanya font tertentu dalam sistem operasi tertentu, tidak adanya antialiasing dalam lingkungan tertentu)
3. Level menengah: tidak terlalu low level (sehingga terlalu kompleks untuk dipelajari), ataupun high level (sehingga semua sudah ditangani, dan tidak ada yang bisa dipelajari)
4. Tersedia secara gratis dan atau open source

Alasan Penggunaan Java

Lingkungan DOS sebenarnya sangat baik digunakan untuk memahami pembangunan grafik secara low level, namun usaha akan banyak diperlukan, dari sekedar menggambar titik, sampai berurusan dengan interrupt untuk mengakses device seperti mouse. Device-device yang lebih kompleks, seperti digital camera, sangat sulit untuk bisa diakses.

Lingkungan Windows memberikan fasilitas yang sangat banyak untuk melakukan pemrograman grafik, baik 2D maupun 3D, namun umumnya pemrograman grafik di Windows membuat kita tidak bisa melihat bagaimana algoritma sebenarnya bekerja, karena terlalu banyak hal yang sudah disediakan oleh Windows.

Pemrograman low level grafik (dengan C/C++) dan mengakses GDI Windows atau Direct X terlalu kompleks sedangkan pemrograman dengan bahasa visual seperti Visual Basic dan Delphi menyembunyikan terlalu banyak hal yang seharusnya dipelajari dalam kuliah grafik. Belajar pemrograman grafik di Windows juga akan mengikat kita pada satu platform tertentu, sedangkan banyak aplikasi grafik yang tidak berjalan di Windows.

Java untuk belajar pemrograman Grafik

Java merupakan bahasa yang cross platform, dan sudah menyediakan primitif grafik 2D dan secara opsional grafik 3D. Java cukup mudah dipelajari, dan bisa mengakses mode grafik dalam lingkungan manapun (X Window, GDI Windows, dll).

Java juga sudah digunakan sebagai sarana pembantu dalam banyak mata kuliah, termasuk juga pemrograman sistem terdistribusi dan sistem operasi.

Java juga tersedia gratis, tidak seperti kebanyakan lingkungan pemrograman lainnya.

Lingkungan pengembangan

Untuk mengembangkan aplikasi Java, tidak diperlukan software komersial apapun, sehingga semua orang dapat memakainya dengan Legal. Software minimal yang diperlukan:

- Editor teks (tidak diperlukan bila menggunakan Netbeans IDE yang juga gratis)
- JDK (sebaiknya versi 1.4 atau yang lebih baru)

Kompilasi program Java dapat dilakukan dalam IDE, atau dengan menggunakan perintah command line:

javac NamaFile.java

Untuk menjalankan program java bisa dilakukan melalui IDE atau dengan command line:

java NamaFile

perhatikan bahwa akhiran .java (atau .class) tidak dibolehkan ketika menjalankan program

Mode Grafik dalam Java

Mode grafik dalam Java dapat diaktifkan dengan menggunakan kelas-kelas dalam package java.awt. Dengan membuat sebuah Frame (Windows) maka otomatis mode grafik akan diaktifkan. Di Windows hal ini bisa dilakukan langsung (karena sistemnya berbasis grafik). Sedangkan di Linux atau OS yang lain harus mengaktifkan mode grafik untuk OS tersebut. Di Linux biasanya yang harus diaktifkan adalah X Window

Ekstensi Grafik pada Java

Selain operasi grafik dasar, Java juga mendukung pemrosesan grafik 2D melalui kelas-kelas Java2D dan pemrosesan grafik 3D melalui paket tambahan (bukan standar) Java3D. Kedua hal ini tidak dibahas di dalam diktat ini, namun informasi mengenai hal tersebut dapat dicari dalam dokumentasi yang disertakan.

BAB IV
ALGORITMA GRAFIK DASAR

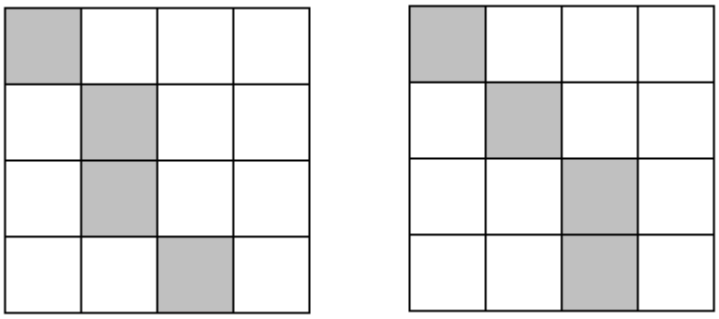
Dalam bab ini akan dibahas mengenai sistem koordinat yang dipakai dalam pembahasan algoritma, cara menggambar garis dengan cara konvensional dan dengan algoritma bresenham, serta cara menggambar lingkaran dengan algoritma konvensional. Dalam bab ini juga akan dibahas mengenai efektivitas penggunakan operasi integer dibanding floating point.

Titik, garis, dan lingkaran merupakan bentuk dasar yang akan dipakai dan merupakan sebagian dari primitif

Sistem koordinat dan ketepatan Piksel

Koordinat layar berlaku seperti matrix, artinya kita tidak bisa secara sembarangan meletakkan titik di sembarang tempat, terkadang kita harus memilih titik mana yang lebih dekat dengan koordinat yang hendak kita gambar. Biasanya koordinat adalah selalu positif dengan titik kiri atas adalah koordinat 0,0, ke arah kanan positif, dan ke arah bawah positif (keduanya semakin membesar). Sumbu X ke arah kanan dan Y ke arah bawah.

Lihat gambar berikut ini, dimana kita hendak menggambar garis dari (0,0) ke (2,3) kita dapat membuat dua pilihan:



Untuk garis yang berbeda dan bentuk yang lebih kompleks, kita perlu membuat lebih banyak pilihan dalam menempatkan piksel.

Algoritma Naif untuk membuat garis

Persamaan sebuah garis dalam matematika adalah:

$Ax + By + c = 0$

Atau:

Persamaan ini mengasumsikan adanya titik-titik dalam koordinat real, dan bukan integer, jadi untuk membuat garis dengan pendekatan ini, kita perlu

Algoritma Naif untuk membuat Lingkaran

Sebuah lingkaran adalah kumpulan titik-titik yang memiliki jarak yang sama dari sebuah titik pusat. Jarak antara sebuah titik sembarang dengan sebuah titik pusat disebut dengan jari-jari lingkaran. Persamaan sebuah lingkaran adalah:

$X^2 + y^2 = r^2$

Algoritma bresenham untuk membuat garis

Jack E. Bresenham adalah professor di Universitas Winthrop menemukan algoritma untuk membuat garis dan lingkaran secara efisien untuk plotter yang ada di IBM pada tahun 1962. Hingga kini algoritma tersebut masih digunakan dan merupakan algoritma paling efisien untuk membuat garis dan lingkaran.

Prinsip algoritma ini sederhana, kita memilih jarak yang terpanjang dalam sumbu X atau sumbu Y, dalam sumbu dengan jarak terpanjang ini pastilah harus ada minimal satu pixel yang digambar di setiap koordinat sumbu tersebut, selanjutnya yang harus dilakukan adalah memilih arah pergerakan di setiap titik.

Misalkan kitaingin menggambar garis pada grid raster (matrix) dimana kita hanya mengijinkan gradien antara 0 sampai dengan 1, inklusif (0 <= m <= 1). Untuk mudahnya kita akan membatasi menggambar garis selalu dengan menambah titik ke arah sumbu x.

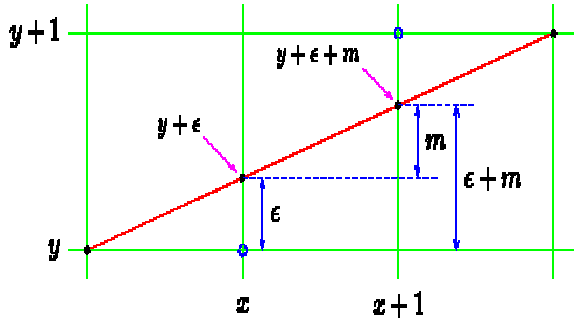
Penyederhanaan ini digunakan untuk mempermudah penjelasan, nanti kita bisa membuat penggambaran untuk gradien sembarang, dan penambahan garis akan dilakukan di sumbu x atau ya tergantung pada panjang garis. Dalam penjelasan ini kita juga akan memakai koordinat kartesian biasa (tidak menggunakan koordinat layar).

Dengan penyederhanaan di atas, maka jika kita telah menggambar di titik (x,y) maka titik berikutnya yang akan digambar kemungkinannya hanya 2:

Di titik (x+1, y) atau
Di titik (x + 1, y + 1)

Jadi untuk menggambar garis pada oktan pertama, kita hanya perlu menentukan dua kemungkinan berikutnya di setiap langkah.

Jika kita gambarkan diagram setelah kita menggambar di titik (x, y), maka akan tampak seperti ini:



Ketika menggambar titik berikutnya di (x,y) kita akan membuat kompromi antara yang sebenarnya ingin kita gambar dengan batasan resolusi layar yang memungkinkan untuk digambari. Kemungkinan besar kita tidak akan bisa tepat menggambar di (x, y) karena titik dengan nilai pecahan tidak bisa digambarkan dengan tepat. Sebagai kompromi kita mengasosiasikan error ε untuk setiap ordinat y, nilai titik yang sebenarnya seharusnya adalah y + ε. Nilai error (ε) ini rangenya dari -0,5 sampai dibawah 0,5.

Ketika bergerak dari x ke x+1 kita menambah nilai ordinat y yang sesungguhnya sejumlah gradien m. Kita memilih untuk menggambar di (x+1, y) jika [erbedaan nilai baru dan y kurang dari 0,5.

$y + \varepsilon + m < y + 0.5$

Selain itu kita menggambar di (x+1, y +1). Sudah jelas bahwa dengan melakukan hal tersebut kita mengurangi kesalahan total antara garis yang sebenarnya secara matematis dengan garis yang tergambar di layar.

Titik yang baru inipun kemungkinan masih memiliki nilai error yang diakumulasi dari error sebelumnya, meskipun sudah dikoreksi jika kita menggambar di (x+1, y +1). Proses ini dapat diteruskan untuk titik x+2 dan seterusnya.

Nilai error yang baru tergantung dari titik yang kita gambar, jika (x+1, y) yang dipilih maka nilai error yang baru adalah

$\varepsilon baru \leftarrow (y + \varepsilon + m) - y$

dan selain itu:

$\varepsilon baru \leftarrow (y + \varepsilon + m) - (y+1)$

Ini memberikan algoritma baru untuk DDA yang menghindari operasi pembualatan, tapi menggunakan variabel error ε untuk mengendalikan penggambaran garis.

$\varepsilon \leftarrow 0$, $y \leftarrow y1$

for x← x1 to x2 do

    plot point at (x, y)

    if ($\varepsilon + m < 0.5$)

    $\varepsilon \leftarrow \varepsilon + m$

    else

$y \leftarrow y + 1$, $\varepsilon \leftarrow \varepsilon + m - 1$

endif

endfor

Tapi algoritma di atas masih menggunakan nilai floating point. Tapi jika kita mengalikan persamaan yang sudah dipakai di atas dengan Δx lalu dengan 2:

$\varepsilon + m < 0.5$

karena $m = \Delta y/\Delta x$ maka

$\varepsilon + \Delta y/\Delta x < 0.5$

$2 \varepsilon \Delta x + 2\Delta y < \Delta x$

Sekarang semua nilainya sudah merupakan bilangan bulat. Jika kita substitusikan ε' untuk $\varepsilon \Delta x$ maka :

$2 (\varepsilon' + \Delta y) < \Delta x$

Sekarang perbandingan ini hanya memakai integer untuk menentukan titik mana yang akan digambar. Karena kita sudah memperbaiki perbandingan untuk pengujian, maka kita juga perlu memperbaiki cara mengupdate nilai ε'. Versi aslinya adalah:

$\varepsilon' \leftarrow \varepsilon + m$

$\varepsilon' \leftarrow \varepsilon' + m - 1$

jika kita kalikan dengan Δx, hasilnya adalah

$\Delta x \leftarrow \varepsilon \Delta x + \Delta y$

$\varepsilon \Delta x \leftarrow \varepsilon \Delta x + \Delta y - \Delta x$

Jika kita melakukan substitusi dengan ε'

$\varepsilon' \leftarrow \varepsilon' + \Delta y$

$\varepsilon' \leftarrow \varepsilon' + \Delta y - \Delta x$

Dengan menggunakan nilai "error" yang baru ini (ε') dengan persamaan perbandingan yang baru dan persamaan update yang baru, kita bisa membuat algoritma Bresenham yang memakai integer saja:

ε′ ← 0, y ← y1

for x ← x1 to x2 do

plot point at (x,y)

if (2 (ε′ + Δy)< Δx)

ε′ ← ε′ + Δy

else

y ← y + 1, ε′ ← ε′ + Δy – Δx

endif

endfor

Sifat algoritma ini

- Hanya memakai integer, sehingga efisien dan cepat
- Perkalian dengan dua bisa diimplementasikan dengan left shift (instruksi shl di pascal atau << di C, C++, Java, dan bahasa lain turunan C)
- Versi ini hanya menangani gradien positif (0<= m <=1) di oktan pertama

Jika diimplementasikan dalam C++, algoritma di atas menjadi:

```
void linev6(Screen &s,
          unsigned x1, unsigned y1,
          unsigned x2, unsigned y2,
          unsigned char colour )
{
  int dx  = x2 - x1,
      dy  = y2 - y1,
      y   = y1,
      eps = 0;

  for ( int x = x1; x <= x2; x++ )  {
    s.Plot(x,y,colour);
    eps += dy;
    if ( (eps << 1) >= dx )  {
      y++;  eps -= dx;
    }
  }
}
```

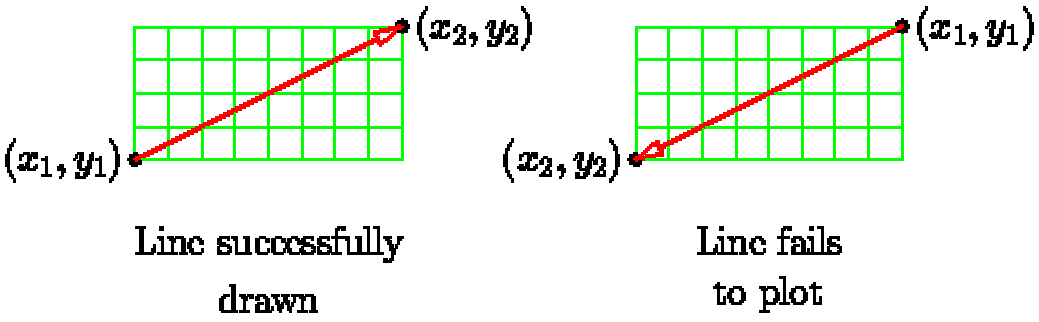
Fungsi ini hanya memakai bilangan bulat, memakai shift untuk perkalian dan mengeliminasi operasi redundan dengan variabel eps.

Implementasi di atas tidaklah lengkap, tidak mengecek argumen yang tidak valid. Implementasi yang benar harus bisa menerima garis dengan gradien berapapun.

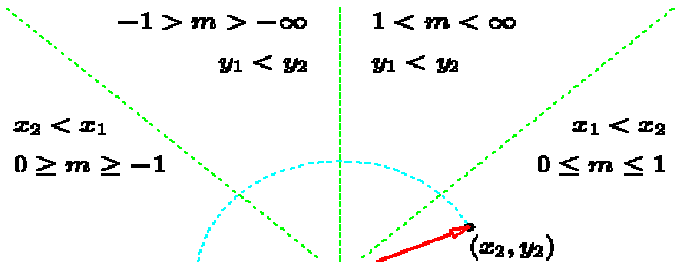
Menangani gradien lain

Jika kita mencoba implementasi C++ di atas kita akan menemukan beberapa kejanggalan. Seperti yang telah dijelaskan, algoritma tersebut gagal untuk gradien negatif, dan positif lebih dari 1. Anda bisa mencobanya untuk melihat hasilnya.

Kemudian kejanggalan berikut adalah bahwa titik awal x1 haruslah lebih kecil dari x2. Jika kita punya dua grais dengan titik ujung yang sama (dengan awal dan akhir yang berbeda, bolak-balik). Maka yang satu berhasil digambar dan yang satu tidak:



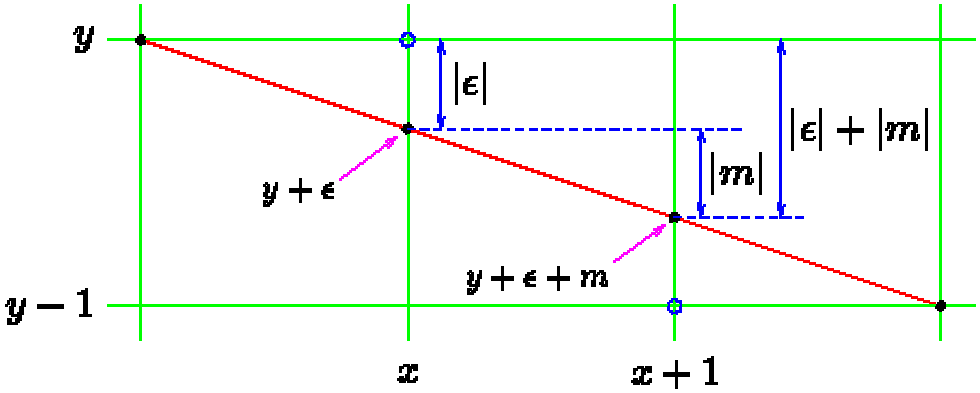
Tapi ini sebenarnya tidak terlalu mengagetkan karena algoritma di atas hanya menambah nilai X. Tapi hal ini juga menunjukkan bahwa algoritma ini menggambar vektor, sehingga arah menjadi pentng. Dengan melihat sebuah vektor dari (x1, y1) sampai (x2, y2) kita menemukan bahwa ada 8 wilayah (atau oktan) dan algoritma di atas hanya bekerja untuk salah satu oktan.



dalam bidang di mana x2 lebih kecil dari x1 bisa ditangani dengan menukarkan titik awal dan titik akhir. Dan tentu saja kita perlu kode untuk menangani gradien besar dengan menambah nilai y alih-alih x.

Namun kita harus berhati-hati karena diagram menunjukan bahwa versi yang telah kita buat tidak bisa menangani gradien negatif (baik untuk m >-1 ataupun m yang lebih kecil), kita harus mengubah algoritma Bresenham untuk menangani kasus ini. (kita akan berurusan dengan m>-1 lalu kita adaptasi untuk m<-1 dengan menambah nilai y alih-alih x).

Bresenham untuk gradien negatif



Andaikan kita menggambar garis dengan gardien antara 0 dan 1 (yaitu gradien negatif kecil. Diberikan suatu titik (x, y) maka pilihan berikutnya adalah menggambar di *(x+1,y-1)* and *(x+1,y)*.
Seperti dalam kasus sebelumnya, akan ada error ε yang diasosiasikan dengan ya. Pilihan untuk menggambar akan didasarkan pada upaya untuk meminimalkan error, sehingga gambarlah di (x+1, y) jika

$$y - (y + \varepsilon + m) < 0.5$$

Selain itu gambar di (x + 1, y -1). Perhatikan bahwa error yang dihasilkan di atas adalah **negatif**, sehingga kita bisa ubah menjadi

$$\varepsilon + m > -0.5$$

Anda bisa bandingkan ini dengan yang dipakai di gradien positif. Aturan untuk mengupdate error juga berbeda untuk gradien negatif:
Jika menggambar di (x + 1, y), nilai error baru adalah:

$$\varepsilon baru \leftarrow (y + \varepsilon + m) - y$$

$$\leftarrow \varepsilon + m$$

Selain itu, menggambar di (x + 1, y – 1) memberikan nilai error yang baru:

$$\varepsilon baru \leftarrow (y + \varepsilon + m) - (y - 1)$$

$$\leftarrow \varepsilon + m + 1$$

Pseudocode untuk algoritma di atas bisa dituliskan sebagai:

```
ε ← 0, y ← y1
for x← x1 to x2 do
        plot point at (x, y)
        if (ε + m > -0.5)
        ε ← ε + m
        else
        y ← y - 1, ε ← ε+ m -1
        endif
endfor
```

Algoritma ini masih memakai nilai floating point. Tapi untuk mengubah algoritma ini supaya hanya memakai integer saja merupakan hal yang mudah (ikuti contoh di atas untuk kasus gradien positif).

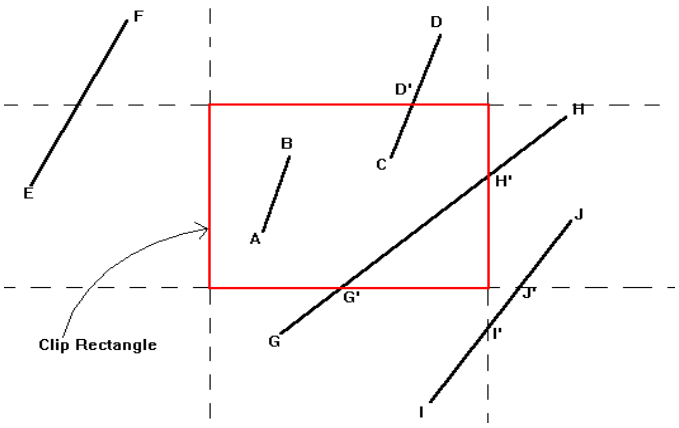
BAB V
CLIPPING DAN ALGORITMA FILL

Dalam bab ini akan dibahas dua buah algoritma penting dalam grafika komputer yaitu clipping dan Fill. Penggambaran garis (dan tentu saja poligon) dapat dioptimasi lebih jauh lagi jika kita mengetahui batasan area yang boleh digambari (clipping area), ada beberapa algoritma yang bisa dipakai untuk melakukan clipping dengan efisien.

Algoritma Fill merupakan algoritma untuk memberi warna pada suatu bidang polygon pada bidang datar. Algoritma ini penting untuk membuat gambar yang memiliki warna atau pola fill.

Clipping

Perhatikan gambar di bawah, kotak merah adalah wilayah yang bisa digambar, dapat dilihat bahwa menggambar garis EF tidak masuk akal, karena tidak akan tergambar. Demikian juga bagian dari garis garis tertentu tidak akan tergambar (yaitu GG', DD', HH' dan IJ). Hal ini akan lebih parah jika ternyata garis yang tidak tergambar tersebut sangat panjang.



Algoritma Clipping yang naif tentu saja dapat diimplementasikan secara sederhana dengan menggunakan algoritma yang memanfaatkan floating point. Algoritma sederhana tersebut adalah:

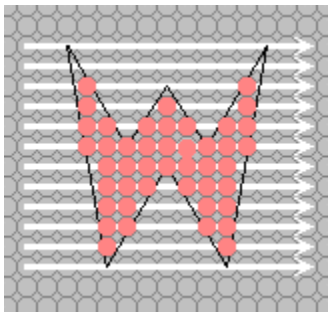
6. Tentukan apakah garis benar-benar ada di luar area penggambaran (seperti garis EF), hal ini bisa dilakukan dengan membandingkan koordinat (x1, y1) dan (x2, y2) dengan koordinat clip, jika garis ada di luar area penggambaran, maka tidak perlu digambar
7. Tentukan apakah seluruh garis ada di dalam area penggambaran dengan cara yang sama dengan cara pertama, jika ya, maka gambar semua garis
8. Selain dua kasus di atas, tentukan titik perpotongan garis dengan batas clip menggunakan persamaan grafik biasa dengan mensubstitusi nilai x dan atau y

Algoritma ketiga tersebut membutuhkan perhitungan floating point.

Fill

Algoritma pertama untuk fill dinamakan algoritma parity fill. Algoritma ini sangat sederhana:

1. proses raster satu barus, atau scanline setiap waktu
2. harus memulai setiap scan line dari suatu titik di lur gambar
3. sambil memproses setiap kolom, state berubah-ubah dari di dalam dan di luar gambar yang difill
4. ketika berada di dalam, ganti nilai piksel dengan warna fill



Kelemahannya adalah

1. hanya menangani gambar yang terisolasi
2. hanya bekerja untuk gambar ideal
3. garis yang sejajar dengan arah scan harus ditangani sebagai kasus khusus

4. tidak bekerja baik untuk gambar yang memiliki perpotongan dengan dirinya sendiri (self-intersecting figures, seperti gambar bintang dibawah)

Dengan segala keterbatasannya, algoritma ini cukup populer untuk beberapa aplikasi, seperti pembuatan font, dan penggambaran poligon sederhana. Beberapa alasannya adalah:

5. Cepat dan cocok untuk implementasi hardware

6. Ada banyak aplikasi yang tidak membutuhkan self-intersecting figures

7. Garis paralel tidak sering ditemui, dan bisa diabaikan (hasilnya masih bisa diterima).

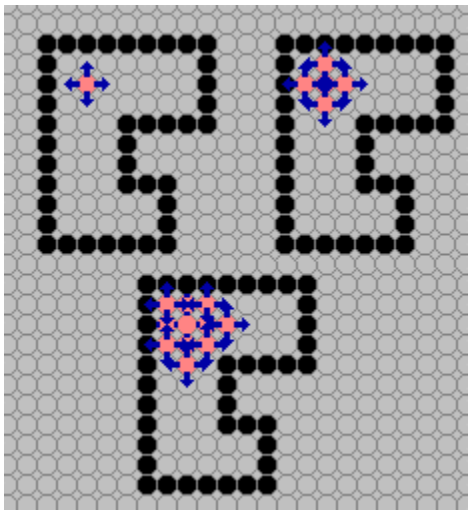
Cara lain untuk melakukan area-fill adalah dengan memulai dari sebuah titik yang kita ketahui berada di dalam gambar dan mengisinya dari dalam keluar. Menggunakan teknik ini maka artis grafik bisa menggambar outline gambar dan memilih warna atau pola dari menu dan mengisi outline tersebut. Proses fill dilakukan ketika sebuah titik di dalam gambar dipilih. Cara seperti ini dapat ditemui di hampir semua program grafik.

Salah satu algoritma untuk melakukan area fill adalah boundary-fill algorithm. Algoritma ini perlu argumen (parameter) berupa koordinat titik awal, warna fill, dan warna background

```
public void boundaryFill(int x, int y, int fill, int boundary)
{
    if ((x < 0) || (x >= raster.width)) return;
    if ((y < 0) || (y >= raster.height)) return;
    int current = raster.getPixel(x, y);
    if ((current != boundary) & (current != fill)) {
        raster.setPixel(fill, x, y);
        boundaryFill(x+1, y, fill, boundary);
        boundaryFill(x, y+1, fill, boundary);
        boundaryFill(x-1, y, fill, boundary);
        boundaryFill(x, y-1, fill, boundary);
    }
}
```

Perhatikan bahwa algoritma ini sifatnya rekursif. Setiap kali kita memanggill boundaryFill, prosedur tersebut dapat memanggil dirinya berkali-kali.

Logika algoritma ini sederhana. Jika kita tidak berada di batas, dan sudah mewarnai, maka kita harus membuat titik di wilayah tetangga, dan tetangga tersebut diminta untuk mengisi tetangganya (rekursif).



Bentuk lain dari area fill adalah kita ingin mengganti warna semua titik yang sama dengan titik saat ini dengan warna baru. Algoritma ini dinamakan flood-fill. Algoritma ini sangat mirip dengan boundary fill, bedanya hanya pada pengecekan warna saat ini saja

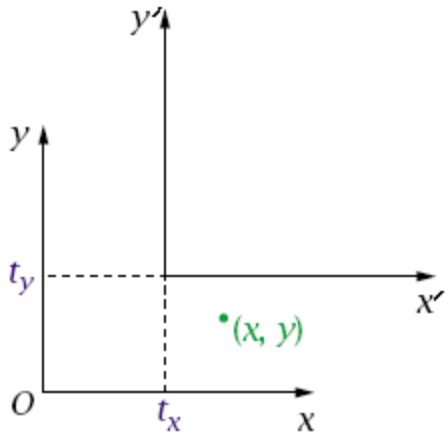
```
public void floodFill(int x, int y, int fill, int old)
{
    if ((x < 0) || (x >= raster.width)) return;
    if ((y < 0) || (y >= raster.height)) return;
    if (raster.getPixel(x, y) == old) {
        raster.setPixel(fill, x, y);
        floodFill(x+1, y, fill, old);
        floodFill(x, y+1, fill, old);
        floodFill(x-1, y, fill, old);
        floodFill(x, y-1, fill, old);
    }
}
```

BAB VI
OPERASI MATRIX UNTUK GRAFIK 2D

Untuk memanipulasi grafik, kita bisa menggunakan operasi matrix. Operasi yang ada meliputi: rotasi, penskalaan, refleksi, shear, dan translasi. Semua operasi ini merupakan operasi dasar dalam membuat aplikasi yang kompleks.

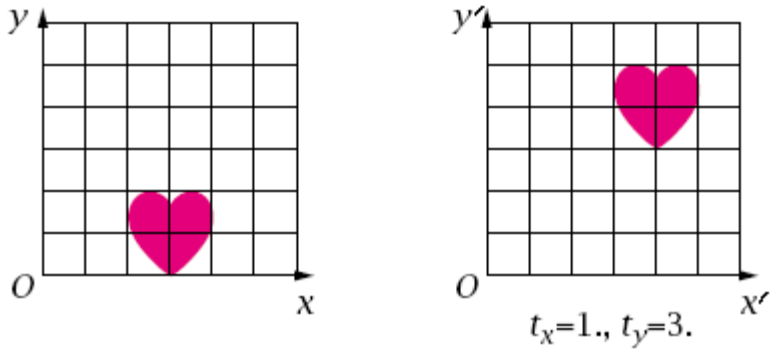
Translasi Koordinat

Translasi koordinat adalah memindah koordinat dari titik (x,y) ke (x', y'), proses ini dilakukan dengan menambahkan suatu nilai dari koordinat:



x' = x + tx
y' = y + ty

Ini adalah contoh translasi untuk objek (bukan titik):

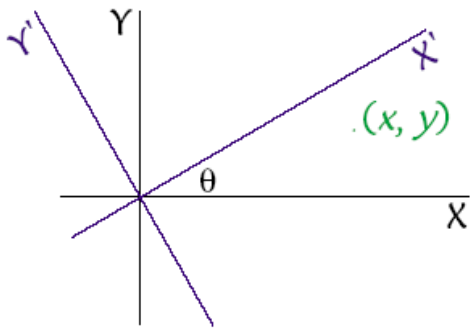


Kegunaan translasi objek umumnya untuk animasi, dimana objek dipindahkan posisinya secara bertahap untuk mendapatkan kesan bahwa objek bergerak.

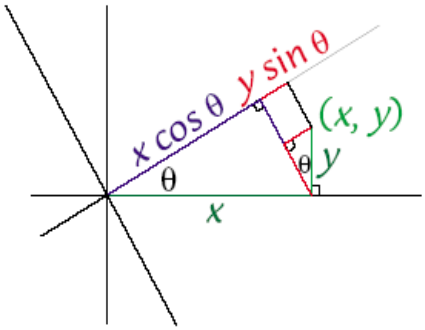
Diantara semua bentuk transformasi, transformasi ini adalah yang paling sederhana.

Rotasi Koordinat

Rotasi koordinat didefinisikan sebagai perubahan koordinat terhadap suatu titik pusat sebesar sudut tertentu. Jika digambarkan:



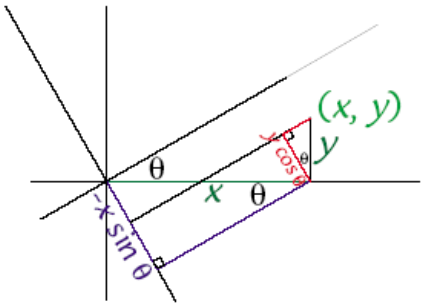
Secara matematis perubahan koordinat X adalah:



Atau:

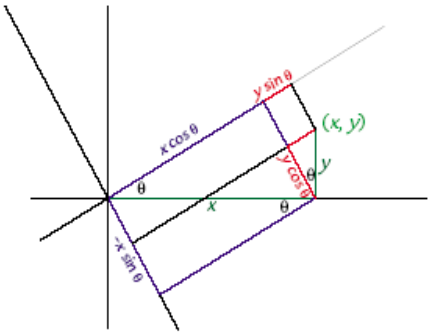
x' = x cos (theta) + y sin (theta)

Sedangkan perubahan koordinat Y adalah



y' = -x sin (theta) + y cos (theta)

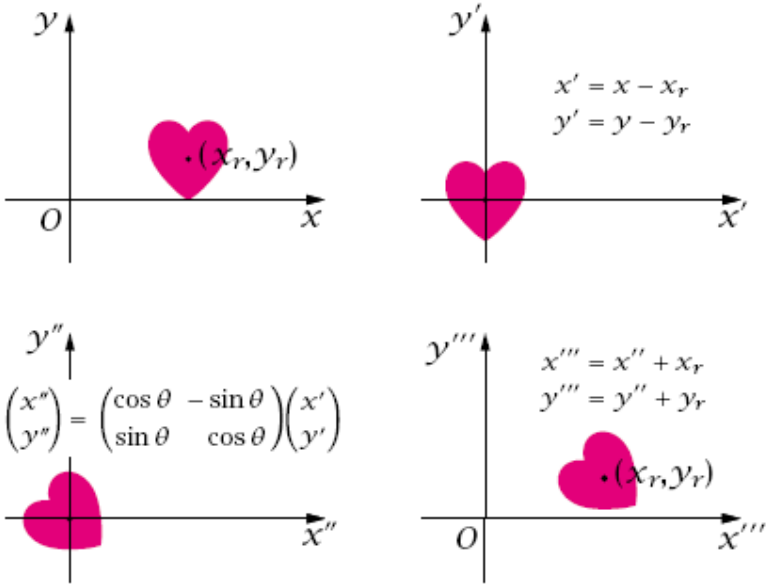
Sehingga rotasi koordinat secara keseluruhan adalah:



Atau jika dinyatakan dalam perkalian matrix:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

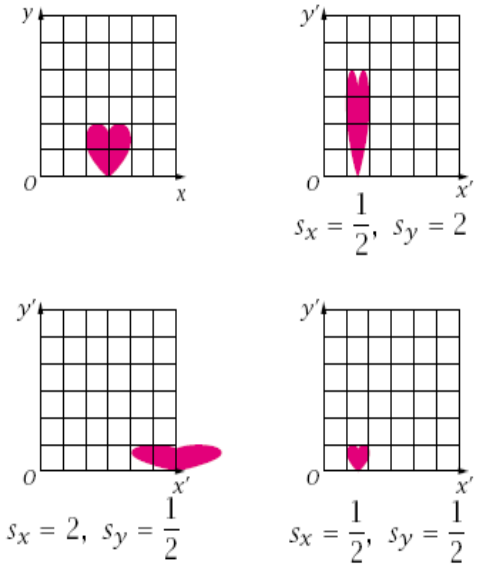
Lalu bagaimana caranya merotasi titik terhadap titik lain? Misalnya kita ingin membuat gambar bulan yang mengelilingi bumi sementara bumi mengelilingi matahari. Jika matahari ada di titik (0,0), maka kita perlu beberapa langkah untuk merotasi bulan terhadap bumi.



Pertama, translasikan objek ke titik (0,0), kedua lakukan rotasi dengan rumus yang telah disebutkan, dan ketiga: kembalikan objek ke tempat semula dengan translasi.

Penskalaan Objek

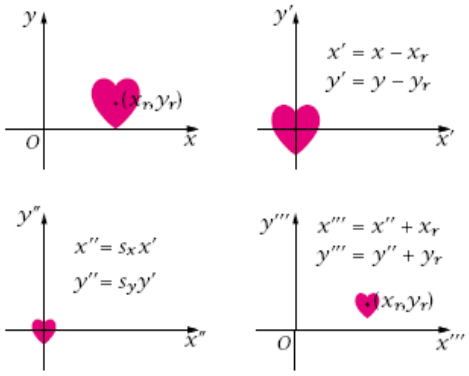
Penskalaan adalah proses untuk memperbesar atau memperkecil objek berdasarkan skala tertentu.



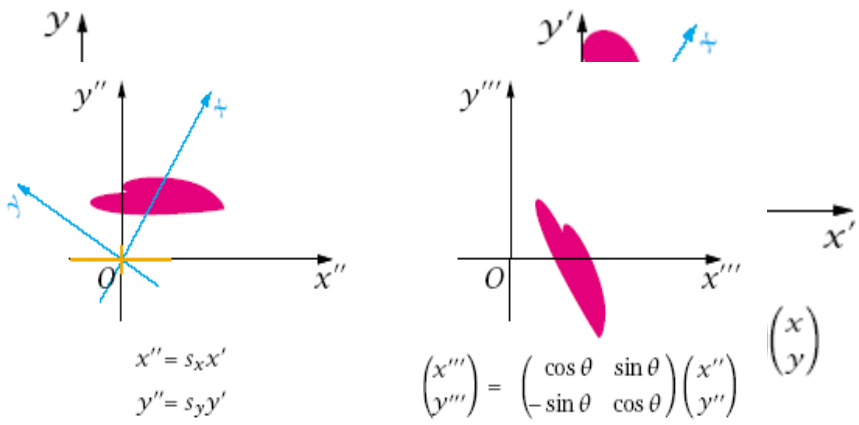
Secara matematis prosesnya adalah:

$$x' = s_x x$$
$$y' = s_y y$$

Sama halnya dengan rotasi terhadap suatu titik selain (0,0), penskalaan juga bisa dilakukan terhadap titik lain:



Penskalaan tidak hanya bisa dilakukan terhadap sumbu X dan Y, tapi terhadap sudut manapun juga, dan efeknya cukup menarik. Inti dari proses ini adalah dengan memutar dulu objek, dan baru melakukan proses penskalaan:



Koordinat Homogen

Andaikan h adalah suatu bilangan real, maka titik manapun dalam koordinat dapat direpresentasikan sebagai tripel homogen:

(xh, yh, h) = (hx, hy, h)

Yang sama saja dengan x = xh/h dan y = yh/h

Perhatikan bahwa xh dan yh hanya digunakan dalam perhitungan, tapi yang ditampilkan tetap koordinat x dan y.

Dengan koordinat homogen, rotasi, translasi, dan penskalaan cukup direpresentasikan sebagai perkalian matrix saja.

Translasi menjadi:

$$\begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$$

$$T(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Rotasi menjadi:

$$\begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$$

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Penskalaan menjadi:

$$\begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$$

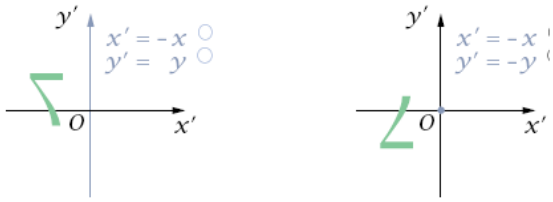
$$S(s_x, s_y) = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Beberapa teorema penting:

$$T(\alpha_1, \beta_1)T(\alpha_2, \beta_2) = T(\alpha_1 + \alpha_2, \beta_1 + \beta_2).$$
$$R(\theta_1)R(\theta_2) = R(\theta_1 + \theta_2)$$
$$S(\alpha_1, \beta_1)S(\alpha_2, \beta_2) = S(\alpha_1\alpha_2, \beta_1\beta_2)$$

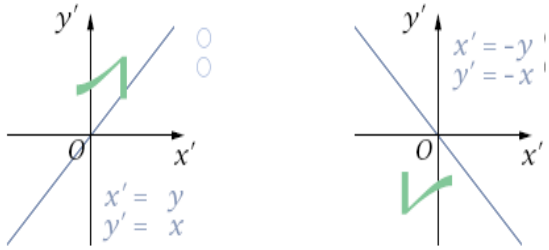
Refleksi

Pencerminan adalah proses untuk membuat salinan objek seperti di cermin. Pencerminan dilakukan terhadap suatu garis, titik yang dekat dengan garis akan memiliki jarak yang sama di sisi garis yang lain.



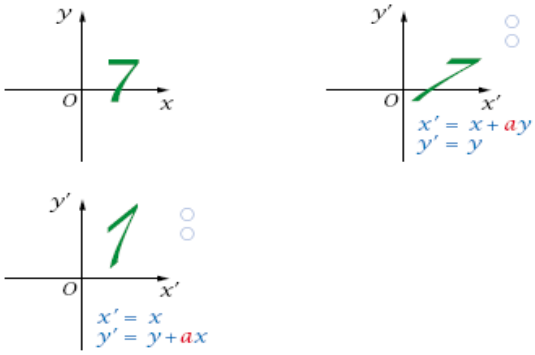
$$\begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}, \qquad \begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$$

Pencerminan tidak hanya dapat dilakukan terhadap sumbu X dan Y



$$\begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}, \qquad \begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$$

Shearing



$$\begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}; \qquad \begin{pmatrix} x'_h \\ y'_h \\ h' \end{pmatrix} = \begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}.$$

Keseluruhan transformasi yang dapat dinyatakan dalam perkalian matrix disebut dengan transformasi Affine.

BAB VII
OPERASI MATRIX UNTUK GRAFIK 3D

Sama dengan operasi yang kita lakukan terhadap grafik2D, kita juga bisa melakukan operasi serupa untuk grafik 3D. Untuk memanipulasi grafik, kita bisa menggunakan operasi matrix sama seperti grafik 2D, tapi koordinat homogen yang digunakan adalah 4 dimensi (matrix yang dipakai menjad 4 x 4). Operasi yang ada meliputi: rotasi, penskalaan, refleksi, shear, dan translasi. Semua operasi ini merupakan operasi dasar dalam membuat aplikasi yang kompleks.

Beberapa operasi tetap sama (hanya diubah menjadi 3 dimensi) sedangkan beberapa operasi lain perlu diubah sedikit definisinya. Misalnya: pencerminan pada bidang 3D harus dilakukan terhadap bidang (dua dimensi), dan bukan garis.
Dalam bagian ini yang akan dibahas hanya operasi dasar saja.

Operasi-operasi tingkat menengah atau lanjut tidak dibahas, misalnya di sini tidak dibahas mengenai rotasi objek terhadap suatu titik sembarang. Rumus untuk hal ini bisa diturunkan dari rumus yang diberikan ditambah dengan contoh pada operasi koordinat 2D.

Koordinat 3D

Koordinat dalam 3 Dimensi direpresentasikan dalam vektor R4. Untuk suatu nilai h yang tak nol maka titik (x, y, z) dalam R3 direpresentasikan sebagai:

$$(hx, hy, hz, h)^T \in \mathbb{R}^4.$$

dalam koordinat homogen

Dan kebalikannya, titik dalam koordinat homogen:

$$(x', y', z', h')^T \in \mathbb{R}^4$$

Bersesuaian dengan titik:

$$\left(\frac{x'}{h'}, \frac{y'}{h'}, \frac{z'}{h'}\right)^T \in \mathbb{R}^3$$

Dalam sistem koordinat dunia.

Jika

$$: \mathbf{x} = (x, y, z, h) \in \mathbb{R}^4$$

Adalah suatu titik dalam koordinat dunia dan:

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

adalah matrix translasi, maka:

$$\mathbf{x}' = T(t_x, t_y, t_z)\mathbf{x}$$

Merupakan translasi titik X terhadap matrix T. Dengan koordinat x bergeser sebanyak tx, y sebanyak ty dan z sebanyak tz

Rotasi

Dalam koordinat dua dimensi, rotasi dilakukan dalam sumbu x dan y, dalam dimensi yang lebih tinggi, maka sumbu rotasi perlu ditentukan.

Rotasi dalam sumbu Z direpresentasikan dalam matrix rotasi-z

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

Demikian juga halnya dengan rotasi dalam sumbu yang lain
Sumbu X:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

Demikian juga dengan:

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Penskalaan

Penskalaan dalam tiga dimensi sama dengan dalam dua dimensi. Matrix untuk proses penskalaan adalah:

$$S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dengan sx, sy, dan sz merupakan penskalaan untuk sumbu x, y dan z. D

$$\mathbf{x}' = S(s_x, s_y, s_z)\mathbf{x}.$$

Shearing

Proses Shearing juga sama di koordinat 3 dimensi, untuk shear x-y, matrixnya adalah:

$$SH_{xy}(sh_x, sh_y) = \begin{pmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Dan operasi yang dilakukan adalah:

$$\mathbf{x}' = SH_{xy}(sh_x, sh_y)\mathbf{x}.$$

Jika h = 1 maka kita memperoleh:

BAB VIII
PROYEKSI GRAFIK 3D PADA BIDANG 2D

Monitor, printer, dan alat-alat yang digunakan untuk menampilkan citra saat ini hanya bisa menampilkan objek 2 dimensi. Hal ini berarti kita perlu cara untuk memproyeksikan objek yang kita buat dalam 3 dimensi ke bidang 2 dimensi.

Koordinat 3D

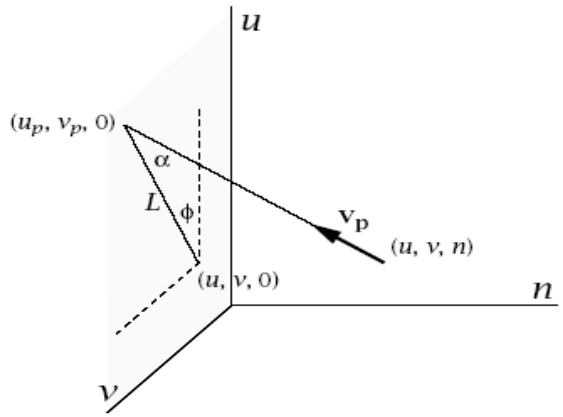
Koordinat 3D direpresentasikan dalam (x, y, z). Sedangkan koordinat 2D direpresentasikan dalam (x, y). Proses untuk mengubah koordinat 3D ke 2D ini disebut dengan proyeksi. Proyeksi bisa dilakukan jika kita mengetahui:

- letak dan arah kamera
- lebar jarak pandang kamera

Untuk hal yang pertama, kita harus menentukan, sedangkan untuk lebar kamera, biasanya areanya adalah seluas layar (meskipun tidak harus seperti itu).

Proyeksi Parallel

Diberikan suatu vektor vp, buat suatu garis yang paralel ke vp melalui setiap titik dalam koordinat view. Perpotongan garis ini dengan bidang penglihatan menjadi hasil proyeksi.



Bentuk yang paling sederhana dari proyeksi ini adalah proyeksi ortografik atau isomorpik, dimana kita hanya menghilangkan nilai z pada koordinat (x, y, z). Hal ini sudah dilakukan sejak jaman purba, seperti dilakukan di zaman paleolitik

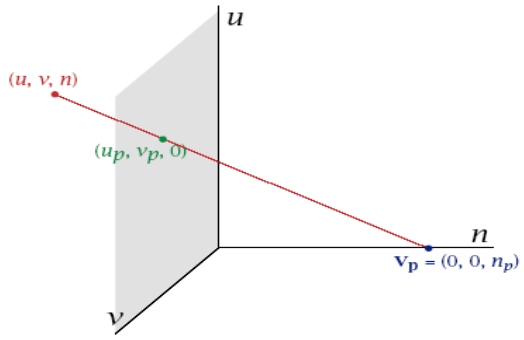
Secara lebih rumit, proyeksi ini bisa dilakukan pada sudut tertentu, namun jenis proyeksi ini tetap tidak menampilkan gambar seperti yang diharapkan oleh mata manusia. Proyeksi yang sering dipakai adalah perspective projection.

Proyeksi Perspective

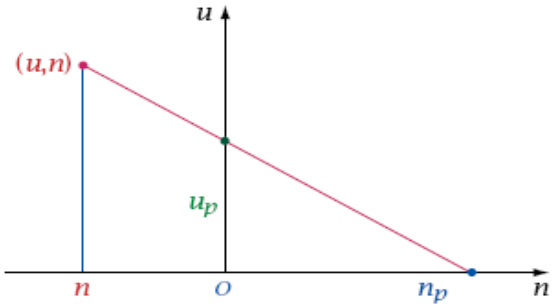
Proyeksi perspective tidak memperhatikan fakta bahwa dalam pandangan kita, benda yang jauh akan terlihat lebih kecil, dan benda yang dekat akan terlihat lebih besar. Hal ini bisa kita lihat pada gambar di bawah. Hal ini dipelopori pada masa renaissance oleh Fillipo Brunelleschi (1377–1446), Leono Battista Alberti (1404–1472), dan Piero della Francesca (c.1420–1492)

Ideal Piazza, Piero della Francesca

Setiap koordinat dalam penglihatan (dalam view) diproyeksikan dengan cara membuat garis terhadap suatu titip referensi vprp. Perpotongan garis dengan bidang ini yang menjadi hasil proyeksi.



Atau jika kita sederhanakan dalam bidang 2 dimensi:



$$\frac{u_p}{n_p} = \frac{u}{n_p - n} \Rightarrow u_p = \frac{u}{1 - \frac{n}{n_p}}$$

Sehingga kita bisa mendapatkan matix untuk proyeksi perspective

$$M_{\text{persective}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{n_p} & 1 \end{pmatrix}$$

Dari sini kita bisa mengetahui bahwa operasi proyeksi perspective adalah perkalian antara matrix perspective dengan koordinat 3D.

BAB IX
HIDDEN SURFACE REMOVAL

Hidden surface removal adalah salah satu cara untuk mempercepat pemrosesan grafik3D. Dengan menghilangkan semua permukaan yang tidak terlihat (yang menghadap ke belakang, atau yang tertutup oleh objek lain), maka waktu yang diperlukan untuk memproses sebuah scene menjadi cepat. Hidden surface removal harus dilakukan dengan tepat, artinya kita tidak boleh menghapus permukaan yang seharusnya terlihat, dan harus efisien yang berarti bahwa semua permukaan yang tidak terlihat tidak perlu diproses.

Ada banyak cara untuk melakukan hidden surface removal, di antaranya adalah: Painter's Algorithm, Binary Space Partitioning Trees, dan Z-Buffering. Masing-masing teknik memiliki kelebihan dan kekurangan. Dari semua algoritma tersebut, yang akan dibahas adalah Z-Buffering, karena memberikan kompromi yang baik untuk kecepatan, kemudahan implementasi, dan ketepatan.

Painter's Algorithm

Algoritma ini berjalan cepat, namun sulit diimplementasikan (terutama test untuk menentukan overlaping). Kelemahan lain algoritma ini adalah ketidak mampuannya untuk memproses dengan benar scene yang kompleks.

Binary Space Partitioning Trees

Algoritma ini lebih cepat dibanding algoritma painter, tapi sulit diimplementasikan dan hanya dapat mengurutkan polygon statik (tidak cocok untuk situasi di mana banyak terdapat polygon dinamik seperti dalam game).

Z-Buffering

Z-Buffering adalah teknik HSR dengan mencatat semua koordinat Z setiap titik yang kita letakkan di layar dalam array yang besar. Ketika akan menampilkan koordinat kita membandingkan dengan koordinat lain untuk melihat apakah ada yang menghalangi di depan, jika ada, maka koordinat tersebut tidak perlu digambar.

Sifat algoritma ini:
- waktu yang diperlukan meningkat linier terhadap jumlah polygon
- lebih cepat dari painter's algorithm untuk lebih dari 5000 polygon
- dapat merender scene dengan benar, tidak peduli sekompleks apapun
- sangat mudah diimplementasikan
- butuh banyak memori
- cukup lambat

Memori yang dibutuhkan untuk implementasi adalah sebesar lebar dikali panjang memori layar:

```
typedef long ZBUFTYPE;

ZBUFTYPE *zbuffer;

zbuffer=(ZBUFTYPE *)malloc(sizeof(ZBUFTYPE)*MEMORYSIZE);
```

Pertama kita harus menginisialisasi zbuffer ini dengan suatu nilai yang kecil (sangat jauh)

```
int c;
for(c=0; c<MEMORYSIZE; c++)
  zbuffer[c]=-32767;
```

Pertanyaan berikutnya adalah: bagaimana caranya kita bisa menentukan koordinat z untuk setiap titik? Kita hanya punya koordinat Z untuk titik tertentu dalam polygon. Kita memerlukan inverse dari proyeksi, persamaan proyeksi kita adalah:

$$u = f \cdot x / z$$

dan

$$v = f \cdot y / z$$

di mana u adalah koordinat x di layar minus Xasal dan v adalah koordinat di layar minus Yasal. Persamaan bidang adalah:

$$Ax + By + Cz + D = 0$$

dari dua persamaan pertama, kita mendapat:

$$x = uz / f$$

and

$$y = vz / f$$

Jika kita masukkan persamaan ini dalam persamaan bidang, kita akan mendapat:

$$A(uz / f) + B(vz / f) + Cz = -D$$

kita bisa mengeluarkan z menjadi:

$$z(A(u / f) + B(v / f) + C) = -D$$

Jadi kita bisa menghitung z:

$$z = -D / (A(u / f) + B(v / f) + C)$$

daripada melakukan pembagian untuk setiap pixel, kita sebaiknya menghitung 1/z:

$$1 / z = -(A(u / f) + B(v / f) + C) / D$$

$$1 / z = -(A / (fD))u - (B / (fD))v - C / D$$

untuk pixel yang berurutan, kita cukup menghitung titik awal:

$$1 / z = -(A / (fD))u_1 - (B / (fD))v - C / D$$

lalu untuk setiap pixel kita hanya perlu menambah:

$$-(A / (fD))$$

Kode programnya kira-kira menjadi:

```
#define FIXMUL (1<<20)

int offset=y*MODEINFO.XResolution+x1;
int i=x1-Origin.x, j=y-Origin.y;
float z_,dz_;
ZBUFTYPE z,dz;

//Initialize 1/z value (z: 1/z)
dz_=((A/(float)Focal_Distance)/-D);
z_=((dz_*i)+( (B*j/(float)Focal_Distance) + C) /-D);
dz=dz_*FIXMUL;
z=z_*FIXMUL;
```

Lalu untuk setiap pixel, kita cukup melakukan:

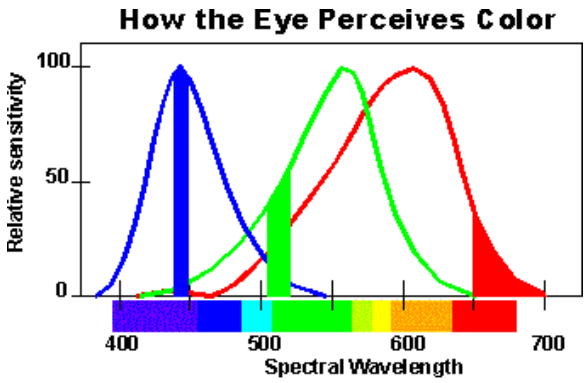
```
if(z>ZBuffer[offset])
 {
    zbuffer[offset]=z;
    SCREENBUFFER[offset]=color;
 }
 z+=dz;
```

BAB X
MANIPULASI WARNA

Warna merupakan bagian penting dalam grafika komputer. Tanpa warna, kita hanya akan berurusan dengan hitam atau putih (piksel digambar atau tidak). Dalam bab ini akan dibahas mengenai warna, bagaimana persepsi mata manusia terhadap warna.

Warna dan mata manusia

Warna adalah karakteristik cahaya yang dapat dipersepsi dan disebutkan berdasarkan namanya. Mata manusia memiliki reseptor untuk 3 warna dalam bentuk sel kerucut (cone cell), yang dapat menerima warna merah, hijau, dan biru. Kepekaan mata terhadap warna ini tidak sama seperti terlihat dalam gambar berikut.



Warna dideskripsikan melalui 3 cara:

- dari namanya

- tingkat kemurnian atau saturasinya
- nilai atau kecerahannya

Istilah Warna

Berikut ini adalah definisi istilah warna (dalam bahasa Inggris karena istilah Indonesianya belum baku):

Chroma: How pure a hue is in relation to gray
Intensity: The brightness or dullness of a hue. One may lower the intensity by adding white or black.
Saturation: The degree of purity of a hue.
Luminance / Value: A measure of the amount of light reflected from a hue. Those hues with a high content of white have a higher luminance or value.

Shade: A hue produced by the addition of black.
Tint: A hue produced by the addition of white.

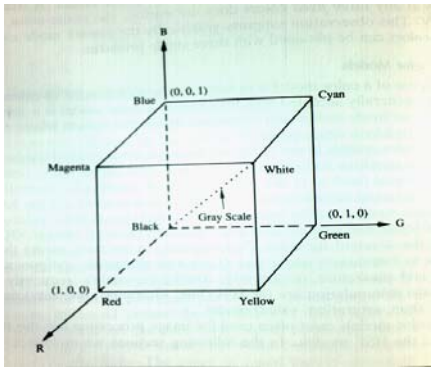
Warna Primer Adalah warna yang tidak bisa dibuat dengan mencampurkan warna lain.
Warna sekunder adalah warna yang merupakan campuran du warna primer

_

Warna Aditif dan Substraktif

Warna aditif adalah sistem warna yang dimulai dari hitam, dan berakhir dengan putih. Artinya untuk membentuk warna putih kita memerlukan semua warna primer. Contoh sistem warna ini adalah RGB yang digunakan di semua monitor dan LCD saat ini.

Model warna RGB juga bisa digambarkan sebagai berikut:



Warna substraktif adalah sistem warna yang dimulai dar putih dan berakhir dengan hitam. Artinya untuk membentuk warna hitam kita perlu mencampurkan

semua warna. Contoh sistem warna ini adalah CYMK yang digunakan oleh pelukis dan juga oleh tinta printer.

Representasi dan Manipulasi Warna di Komputer

Umumnya kita hanya akan berurusan dengan sistem warna RGB karena Sistem Operasi atau layer di atasnya akan mengatur konversi ke sistem CYMK (termasuk juga kalibrasi warna yang diperlukan).

Dalam sistem yang ideal, kita memiliki jumlah bit tertentu untuk warna merah, hijau, dan biru. Dalam sistem yang mendukung 16.7 juta warna (24 bit), ada 8 bit dialokasikan untuk masing-masing warna. Pada sistem yang memiliki jumlah warna kurang dari itu (65K warna = 16 bit, 256 warna 8 bit, 4096 warna 12 bit, 16 warna 4 bit), maka jumlah RGB yang bisa direpresentasikan terbatas.

Dalam sistem yang terbatas, terkadang warna yang bisa dipakai hanyalah dari pilihan tertentu saja (sistem warna palet). Misalnya pada sistem 16 warna CGA, kita hanya bisa memakai 16 warna dalam 1 waktu, dengan sebelumnya kita menentukan dulu nilai masing-masing 16 warna tersebut.

Beberapa manipulasi warna yang bisa dilakukan adalah:

1. mengambil komponen warna tertentu
2. mempergelap atau memperterang gambar (dengan menambah atau mengurangi masing-masing komponen warna)
3. membuat efek transparan dengan menjumlahkan dua warna dengan lebih dulu memberi bobot (warna = bobot_warna1*warna1 + bobot_warna2*warna2) dimana bobot_warna1 bernilai antara 0 sampai 1 dan bobot_warna2 merupakan 1 minus bobot_warna1

Memori Untuk Warna

Memori yang dibutuhkan untuk menangani warna adalah sebesar resolusi layar dikalikan dengan jumlah bit per pixel. Contoh: layar 1024 x 768 24 bit (true color) membutuhkan memori 768 x 1024 x 3 byte = 2304 kb atau 2.25 Mb.

Dari perhitungan bisa dilihat bahwa gambar berwarna memerlukan banyak memori dibanding gambar hitam putih (bisa 32 kali lipat untuk gambar hitam putih murni vs gambar berwarna 32 bit). Pada kasus manipulasi gambar yang bisa diundo misalnya, kita harus menyimpan state gambag sebelumnya, dan tentunya hal tersebut sulit dilakukan di sistem yang tidak memiliki cukup memori.

BAB XI
ILUMINASI DAN SHADING

Setelah bisa membuat dan memanipulasi objek 3D serta memproyeksikan objek tersebut dalam bidang 2D, sekarang saatnya untuk menambah realisme pada objek 3D tersebut. Realisme ini ditambahkan dengan iluminasi dan shading. Iluminasi adalah proses pemberian cahaya pada objek sedangkan shading adalah proses untuk memperhitungkan bayangan objek.

Secara formal, proses iluminasi adalah proses untuk menghitung intensitas titik pada suatu permukaan, sedangkan shading adalah penggunaan intensitas untuk memberi bayangan pada polygon.

Iluminasi

Pencahayaan bisa dilakukan secara lokal atau global. Pada pencahayaan lokal, kita tidak memperhitungkan objek lain di sekitar yang mungkin akan memberikan pengaruh cahaya pada gambar (refleksi, cahaya tambahan), dalam model lokal, seolah-olah setiap benda hanya terpengaruh oleh sumber cahaya.

Pada model global, ada satu sumber cahaya untuk semua objek yang terlihat yang menyinari semua objek. Model ini lebih kompleks karena memperhitungkan refleksi dari objek lain.

Model Iluminasi

Model iluminasi menyatakan komponen cahaya yang dipantulkan atau diterima oleh sebuah permukaan. Ada tiga komponen warna dasar: *ambient*, *diffuse*, dan *specular*.

Ambient Reflected Light

Sumber cahaya yang tidak berarah yang merupakan hasil dari pemantulan berulang dari lingkungan sekitar. Diasumsikan intensitasnya sama untuk semua objek yaitu Ia. Ada suatu konstanta refleksi ambient yang besarnya antar 0 sampai 1 yang menentukan besarnya cahaya ambient yang dipantulkan oleh objek (0 berarti menyerap semua sedangkan 1 berarti memantulkan semua. Koefisien yang dimaksud biasanya merupakan property atau sifat dari suatu bahan.

Persamaan iluminasi ambient adalah:
$I = k_a I_a$

Dimana I adalah intensitas cahaya yang dipantulkan, k adala koefisien dan I adalah intensitas ambient.

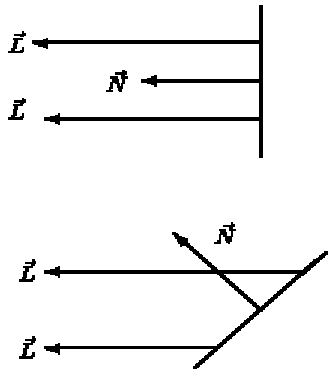
Diffuse Light

Difusi adalah cahaya yang dipantulkan atau dipancarkan dari suatu titik dengan intensitas yang sama ke segala arah. Contoh cahaya difusi yang dipantulkan adalah cahaya dari permukaan polos seperti kertas atau dinding. Contoh cahaya difusi yang direfraksikan adalah dari kaca susu.

Refleksi difusi dimodelkan dengan hukum Lambert yang menyatakan bahwa kecerahan hanya bergantung pada sudut antara arah sumber cahaya dan bidang normal permukaan.

Hukum pertama Lambert:
Iluminasi suatu permukaan yang diterangi oleh cahaya vertikal terhadap sumber cahaya sebanding dengan akar kuadrat antara permukaan dan sumber cahaya.
Hukum kedua Lambert:
Jika cahaya bertemu dipermukaan pada sudut tertentu, maka iluminasinya sebanding dengan kosinus antara sudut itu dengan permukaan normal
Hukum ketiga Lambert:
Intensitas iluminasi berkurang secara eksponensial selaras dengan jarak jika melewati medium yang menyerap.

Gambar berikut menjelaskan mengenai hukum Lambert



sehingga persamaan iluminasinya adalah:

dimana theta adalah sudut antara normal permukaan dan vektor cahaya
Ip adalah intensitas sumber cahaya
Kd adalah koefisien diffuse reflection.

Cahaya Specular

Komponen specular dari refleksi atau refraksi menghasilkan efek highlight akibat refleksi. Efek dari cahaya ini seperti cermin yang memberikan efek mengkilap pada objek. Persamaan untuk cahaya seperti ini diberikan oleh Bui-Thong Phong, pendekatan ini sangat terkenal:

$$I = W(\theta) I_p \cos^n(\phi),$$

2 Shading Method

Flat Shading

Shading ini disebut juga constant shading atau faceted shading, ini merupakan metode shading yang paling sederhana. Aplikasikan persamaan iluminasi untuk setiap polygon. Pendekatan ini valid jika:

- sumber cahaya berada di tempat yang tak hingga jauhnya

- viewer berada di tempat yang tak hingga jauhnya
- poligon merupakan permukaan yang sebenarnya (bukan aproksimasi)

Goraud Shading

Goraud Shading merupakan metode interpolasi intensitas. Intensitas di hitung disetiap vertex di poligon dan diinterpolasi sepanjang busur dan permukaan poligon. Kita bisa menggunakan model polynomial untuk merepresentasikan permukaan yang mulus seperti tabung atau bola.

Phong Shading

Phong Shading merupakan metode shading interpolasi vektor normal. Setiap vektor normal pada busur dihitung seperti pada Goraud shading. Vektor normal diinterpolasi sepanjang busur poligon dan di antara busur pada setiap scanline.

BAB XII
TEXTURE MAPPING

Texture mapping adalah suatu teknik untuk "melapisi" suatu objek 3D dengan suatu citra 2D. Texture mapping memungkinkan kita membuat objek 3D yang cukup realistis berdasarkan polygon yang sederhana. Contohnya kita bisa membuat gambar aneka planet dengan memproyeksikan aneka tekstur planet (misalnya: tekstur biru dengan awan untuk bumi, dan merah untuk mars) terhadap suatu bola.

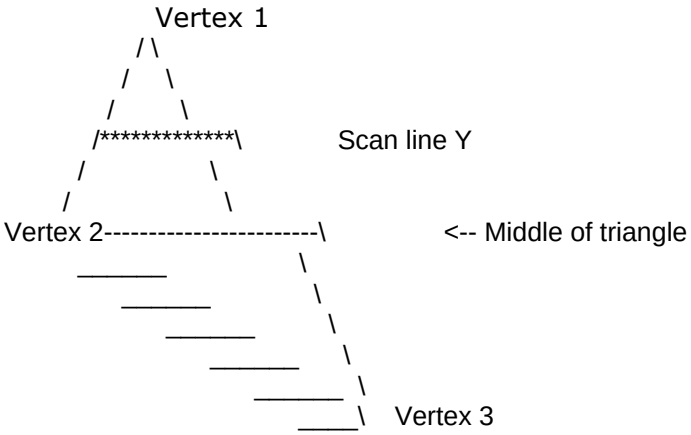
Untuk dapat melakukan texture mapping tersebut citra yang diinginkan harus diproyeksikan kepada objek 3D dengan rumus tertentu. Untuk benda 3D sederhana dengan arah tertentu, proyeksi ini bisa sederhana, namun untuk semua objek 3D secara umum (terutama polygon yang tidak teratur), teknik ini cukup sulit.

Dalam bab ini kita hanya akan belajar untuk melakukan texture mapping terhadap benda sederhana. Untuk benda yang tidak memiliki lengkungan, maka pembuatan tekstur hanya dilakukan terhadap segitiga. Polygon lain bisa dikomposisi dari segitiga (persegi panjang adalah gabungan dua segitiga, trapesium adalah gabungan persegi panjang dengan segitiga, dan seterusnya).

Filling Segitiga

Untuk dapat memahami texture mapping kita harus bisa mengisi sebuah segitiga. Intinya adalah:

- cari ujung kiri dan kanan segitiga
- isi permukaan tersebut



Jadi untuk mengisi segitiga di atas, kita harus mengisi scanline yang ada di bagian atas dari tengah segitiga.

```
For y = y1 to y2 do
    FillScanLine(left[y], right[y], colour)
End For
```

Dan kemudian dari tengah ke simpul ke 3:

```
For y = y2 to y3 do
    FillScanLine(left[y], right[y], colour)
```

End For

Kita asumsikan bahwa left[y] dan right[y] adalah ujung-ujung untuk koordinat Y. Tentu saja kedua loop tersebut dapat dioptimasikan menjadi satu loop, namun untuk kejelasan algoritma, kita akan membiarkan kode tersebut apa adanya.

Mengaplikasikan Texture Mapping

Untuk melakukan texture mapping terhadap segitiga, kita hanya perlu melakukan filling terhadap segitiga, bedanya yang difill bukanlah warna konstan, tapi dari pemetaan terhadap image texture. Karena tekstur bentuknya bukan segitiga, maka kita perlu melakukan transformasi, untuk setiap scanline:

1. Hitung lebar scanline saat ini
2. Hitung lebar Texture dibagi lebar scanline
3. Gunakan nilai tersebut untuk menghitung pixel mana saja yang akan dimap

BAB XIII
ANIMASI

Animasi merupakan teknik untuk membuat gambar bergerak buatan. Berbeda dengan video yang merupakan rekaman dari kejadian di dunia nyata, animasi merupakan sepenuhnya buatan manusia. Animasi pada dasarnya adalah penampilan gambar secara berganti ganti dengan cepat (umumnya 24-30 frame per detik).

Animasi bisa dibuat predefined, artinya setiap frame sudah digambar sebelumnya (oleh seniman), dan hanya ditampilkan secara bergantian, atau dinamis, artinya setiap frame digambar secara dinamis oleh program. Animasi yang statis tidak dibahas dalam bab ini.

Algoritma dasar Animasi
Untuk animasi non interaktif, maka algoritma dasar untuk sebuah animasi adalah sebagai berikut:

```
while (not done) {
        hapus_layar();
        gambar_frame_ke(i);
        delay(n);
        i = i + 1;
}
```

teknik semacam ini memiliki kelemahan, yaitu penggambaran perlu dilakukan dengan menghapus seluruh layar lebih dulu, hal ini lambat dan bisa menyebabkan flicker (kedipan) pada animasi. Cara yang lebih baik adalah dengan menghapus daerah gambar sebesar gambar yang berubah saja. Teknik ini disebut dengan dirty rectangle animation.

Double Buffering
Teknik dirty rectangle memiliki kegunaan terbatas, apabila ternyata gambar yang berubah ada banyak (mendekati atau sebesar ukuran layar, terutama pada animasi dengan banyak gerakan) maka tetap saja seluruh layar perlu digambar ulang sehingga menyebabkan kedipan.

Ada cara untuk menggambar dengan cepat tanpa menimbulkan efek kedipan, yaitu dengan double atau tripe buffering. Pada double buffering, kita menggambar latar belakang, dan gambar baru disebuah kanvas, dan langsung mengcopy isi kanvas tersebut ke layar setelah semua selesai tergambar.

Teknik triple buffering adalah teknik serupa yang memanfaatkan kanvas tambahan yang sudah berisi latar belakang animasi. Jadi teknik yang digunakan adalah: salin latar belakang ke kanvas sementara, gambar kanvas, salin kanvas ke layar. Dengan teknik ini maka penggambaran bisa dilakukan dengan cepat.

Animasi interaktif
Untuk animasi interaktif, algoritma di atas perlu dikoreksi untuk menangani input dari pengguna dan mengubah animasi sesuai dengan input pengguna tersebut.

```
while (not done) {

        proses_input_user();
        hasilkan_frame();
        hapus_layar();
        gambar_frame();
        delay(n);
        i = i + 1;
}
```

Seperti dapat dilihat bahwa algoritma ini tidak menggunakan teknik pemrograman event based. Teknik pemrograman event based terkadang justru menyulitkan dalam animasi interaktif karena kita tidak bisa mengontrol kapan input user bisa diterima atau tidak (bisa saja hal ini dilakukan tapi menambah overhead pemrosesan).