LAPORAN PENELITIAN

Explorasi Java 2D Rendering Engine

Disusun oleh:

Luciana Abednego

Cecilia E. Nugraheni

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI DAN SAINS

UNIVERSITAS KATOLIK PARAHYANGAN

ABSTRAK

Proses *rendering* adalah proses penggambaran primitif-primitif grafik pada piranti keluaran seperti layar monitor atau printer. Proses penggambaran ini mencakup berbagai proses pengaturan berbagai informasi geometri, seperti mengatur arah pandang (*view point*), tekstur, pencahayaan (*lighting*), maupun bayangan (*shading*). Penelitian ini mengeksplorasi sebuah *rendering* engine Java 2D. Hasil penelitian ini dirangkum dalam sebuah modul eksplorasi *rendering* engine Java 2D dan program-program kecil untuk melihat langsung jalannya beberapa metode / algoritma dasar komputer grafik.

DAFTAR ISI

DAFTAR GAMBAR

LAMPIRAN A

BAB I
PENDAHULUAN

1.1. Latar Belakang

Objek 2D dibangun dari primitif-primitif dasar seperti garis, polygon, elips, teks, maupun gambar. Setiap objek geometri tersebut memiliki atribut yang dapat berupa: warna, transparansi, tekstur, dan jenis garis. Objek-objek geometri yang masih berupa model tersebut kemudian digambar pada piranti keluaran melalui sebuah *rendering* engine. Sebuah *rendering* engine mendefinisikan fungsi-fungsi untuk mewarnai, menggambar, melakukan proses transformasi, pemotongan (clipping), dan berbagai fungsi *rendering* lainnya.

Pada penelitian ini, *rendering engine* yang akan dieksplorasi adalah Java 2D. Java 2D (atau sering disebut sebagai "2D API") memperbaiki kelemahan-kelemahan grafik versi sebelumnya. *Rendering* engine Java 2D dapat menghasilkan gambar dengan kualitas profesional pada piranti keluaran seperti monitor atau printer. Pada Java 2D, primitif garis dapat diatur ketebalan maupun jenisnya. Penggambaran suatu primitif grafik dengan pola tertentu, atau teknik lain yang lebih kompleks seperti masalah *image processing*, penanganan huruf, dan sebagainya. Java 2D mampu menghasilkan gambar dan memanipulasi tiga jenis objek grafik: bentuk geometri, teks, maupun gambar.

1.2. Pertanyaan Penelitian

Pertanyaan dalam penelitian ini adalah :

1. Bagaimana konsep grafika komputer 2D?
2. Bagaimana pemodelan Java2D?
3. Bagaimana mengkolaborasikan fungsi-fungsi penggambaran dalam Java2D untuk membuat aplikasi grafik sederhana?

1.3. Tujuan Penelitian

Tujuan dari penelitian ini adalah:

1. Mempelajari konsep grafika komputer 2D.
2. Mempelajari pemodelan Java2D.
3. Mengkolaborasikan fungsi-fungsi penggambaran Java2D untuk membuat aplikasi grafik sederhana.

1.4. Hipotesa

Hipotesa penelitian ini adalah *rendering engine* Java2D mendukung berbagai proses penggambaran grafika 2D.

1.5. Keluaran Penelitian

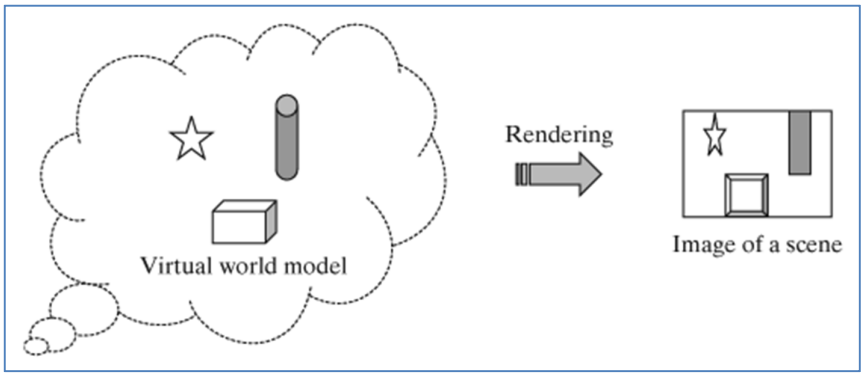
Hasil penelitian ini berupa:

1. Beberapa perangkat lunak grafik sederhana yang dibangun dengan *rendering engine* Java2D.
2. Modul ajar yang berisi penjelasan masing-masing perangkat lunak grafik tersebut.

BAB II
Teori Dasar Grafik

2.1. Grafika Komputer

Grafika komputer mempelajari teori dan teknik pemodelan, pemrosesan, dan penggambaran (*rendering*) objek grafik dalam komputer. Prosesnya adalah dengan membangun model dalam *virtual world,* kemudian berdasarkan model tersebut menggambar *scene* dalam arah pandang tertentu pada piranti keluaran, seperti terlihat pada gambar 2.1. Grafika komputer dapat digunakan di berbagai bidang kehidupan, mulai dari bidang seni, sains, bisnis, pendidikan dan juga hiburan. Beberapa bidang aplikasi spesifik dari grafika komputer di antaranya Antarmuka pengguna (*Graphical User Interface* - GUI), Peta (*Cartography*), Kesehatan, Perancangan objek (*Computer Aided Design* - CAD), Sistem multimedia, Presentasi grafik, Presentasi data saintifik, Pemrosesan citra, Simulasi, Pendidikan, dan sebagainya[1].



Gambar 2.1 Proses *rendering* grafika komputer.

2.2. Rendering

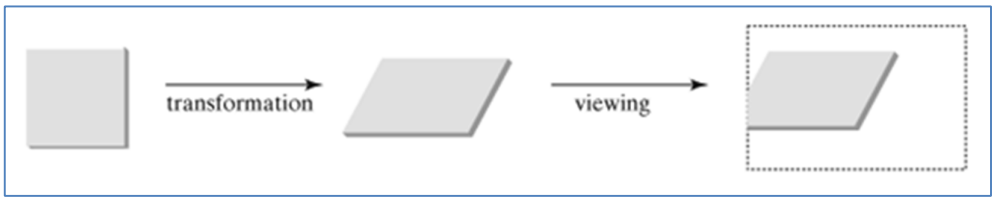
Beberapa definisi megenai proses *rendering*:

[1] What is Computer Graphics?, Cornell University Program of Computer Graphics. *Last updated* 04/15/98. *Accessed* 11/17/09.

1. Proses penggambaran primitif-primitif grafik pada piranti keluaran seperti layar monitor atau printer.
2. Proses menghasilkan gambar dari sebuah model. Model ini menyimpan berbagai informasi geometri, arah pandang (*view point*), tekstur, pencahayaan (*lighting*), maupun bayangan (*shading*).

Proses yang terjadi pada tahap *rendering* adalah sebagai berikut [2]:

Sebuah objek grafik, pertama kali didefinisikan pada sebuah ruang objek (*object space*). Objek ini kemudian ditempatkan pada sebuah "dunia" ruang 2D (disebut dengan istilah *virtual world space*) melalui proses transformasi objek. Proses transformasi objek mengubah bentuk dan lokasi objek. Proses *rendering* (*viewing*) kemudian menangkap "foto" hasil transformasi tersebut pada posisi tertentu. Daerah hasil "foto" ini disebut sebagai ruang pandang (*viewing space*). Proses *viewing* tidak mengubah bentuk objek, hanya mengubah arah pandang pada keseluruhan *world space*. Pada grafika 2D, baik dunia yang akan dimodelkan (*virtual world space*) maupun ruang pandangnya (*viewing space*) keduanya berada dalam bentuk dua dimensi. Gambar berikut memperlihatkan *transformation and viewing pipeline* grafika 2D.



Gambar 2.2 Objek grafik 2D digambar melalui *transformation and viewing pipeline.*

Jika dirangkum, langkah-langkah dasar grafika 2D adalah sbb.:

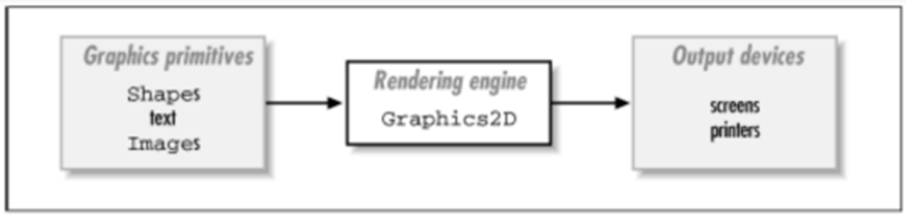
1. Membangun objek 2D
2. Proses transformasi objek
3. Aplikasikan warna dan berbagai properti lain pada objek
4. *Render scene*

Sistem grafik 2D me-*render scene* berdasarkan informasi geometri, transformasi, dan atribut grafis lain.

2.3. Rendering Engine

Primitif-primitif grafik pada virtual world space atau disingkat dengan istilah model, kemudian "dikirimkan" pada *rendering* engine untuk digambar pada piranti keluaran.

Sebuah *rendering* engine mendefinisikan fungsi-fungsi untuk mewarnai, menggambar, melakukan proses transformasi, pemotongan (clipping), dan berbagai fungsi *rendering* lainnya [2]. Gambar berikut memperlihatkan proses *rendering* menggunakan sebuah *rendering engine.*



Gambar 2.3 Rendering engine dalam grafika komputer.

Rendering engine yang akan digunakan pada praktikum ini adalah *class* Graphics2D pada Java. Beberapa fungsi penting pada *class* ini di antaranya adalah:

Paint: menentukan berbagai parameter penggambaran untuk mengisi bentuk geometri.

Stroke: menentukan berbagai parameter penggambaran untuk batas luar objek geometri.

Font: penggambaran teks dilakukan dengan menciptakan suatu bentuk geometri untuk merepresentasikan karakter yang dimaksud.

Transformation: mendefinisikan jenis transformasi yang akan dilakukan pada primitif geometri, seperti translasi, rotasi, penskalaan.

Compositing rule: menentukan kombinasi warna pada area penggambaran.

Clipping shape: menentukan bentuk area penggambaran. *Pixel* lain di luar daerah penggambaran ini akan diabaikan.

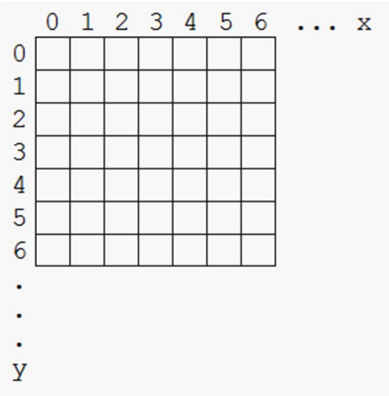
Rendering hints: menentukan teknik penggambaran berbagai primitif grafik.

2.4. Sistem Koordinat Layar

Layar dibagi menjadi kotak-kotak kecil yang tersusun seperti sebuah matriks. Setiap kotak memiliki posisi yang dinyatakan dalam bentuk koordinat kartesius (x,y). Nilai x menyatakan posisi kolom, sedangkan nilai y menyatakan posisi baris. Koordinat layar yang digunakan akan sedikit berbeda dengan koordinat kartesius biasa. Nilai y pada sistem koordinat ini membesar dari atas ke bawah, sehingga posisi koordinat (0,0) terletak pada pojok kiri atas layar [2].

Nilai maksimum dari x dan y dapat berbeda-beda, tergantung dari resolusi layar. Jika sedang berjalan pada resolusi 800x600, maka nilai jangkauan nilai x adalah 0 <= x < 800, dan jangkauan y adalah 0 <= y < 600.

Resolusi layar dapat diubah-ubah sesuai dengan kemampuan monitor dan *graphic card* yang dimiliki suatu komputer. Semakin tinggi resolusi, semakin baik tampilan yang dihasilkan.

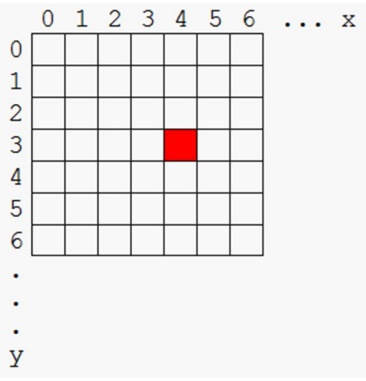


Gambar 2.4 Sistem koordinat layar

2.5. Titik

Titik pada komputer grafik biasa dikenal dengan istilah *pixel* (singkatan dari *picture element*). *Pixel* dapat digambarkan pada layar dengan menentukan: posisi koordinat di layar (x,y) dan warna dari *pixel* tersebut. Menggambar sebuah *pixel* dapat dibayangkan seperti mengisi penuh satu kotak pada posisi tertentu dengan suatu warna [2].

Gambar di bawah ini memperlihatkan titik berwarna merah di posisi (4,3).

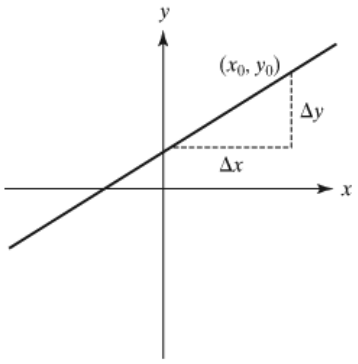


Gambar 2.5 *Pixel* berwarna merah di titik (4,3)

2.6. Garis

Primitif lain yang penting dalam komputer grafik adalah Garis, atau lebih tepatnya Segmen Garis. Garis dapat digambar menggunakan titik-titik yang disusun sedemikian rupa. Sebuah persamaan dapat digunakan untuk menyatakan hubungan antara nilai x dan nilai y pada sistem koordinat. Sebagai contoh, garis pada gambar berikut dapat direpresentasikan dalam persamaan polinomial derajat 1 (persamaan linear).

$$Ax + By + C = 0$$



Gambar 2.6 Segmen Garis

2.7. Lingkaran

Sebuah lingkaran dengan titik pusat di (0,0) dan jari-jari R dapat direpresentasikan dengan persamaan:

$x^2 + y^2 = r^2$

Jika titik pusat lingkaran adalah (x_c, y_c), persamaan menjadi :

$(x - x_c)^2 + (y - y_c)^2 = r^2$

Persamaan ini dapat digunakan untuk menghitung posisi titik-titik luar lingkaran dengan cara mengisi nilai x mulai dari x_c - r sampai x_c + r dengan pertambahan 1 pada setiap langkahnya, lalu menghitung nilai y untuk setiap posisi x tersebut dengan persamaan:

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

2.8. Elips

Sebuah elips yang berpusat di titik (x_0,y_0) dapat direpresentasikan dengan persamaan:

$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1$$

Bentuk persamaan lain yang umum digunakan untuk merepresentasikan bentuk kurva dapat dinyatakan dengan persamaan *parametric*. Dengan menggunakan persamaan *parametric*, selain variabel x dan y, digunakanlah variabel ketiga, t. Kemudian, baik untuk x dan y, masing-masing dinyatakan dalam fungsi dari t:

x=f(t)

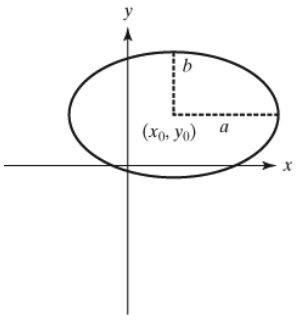
y=g(t)

Keuntungan menggunakan persamaan ini adalah persamaan ini memberikan fungsi evaluasi yang langsung terhadap nilai koordinat.

Sebuah elips, dapat dinyatakan dengan persamaan *parametric*:

$$x = x_0 + a\cos t$$

$$y = y_0 + b\sin t$$



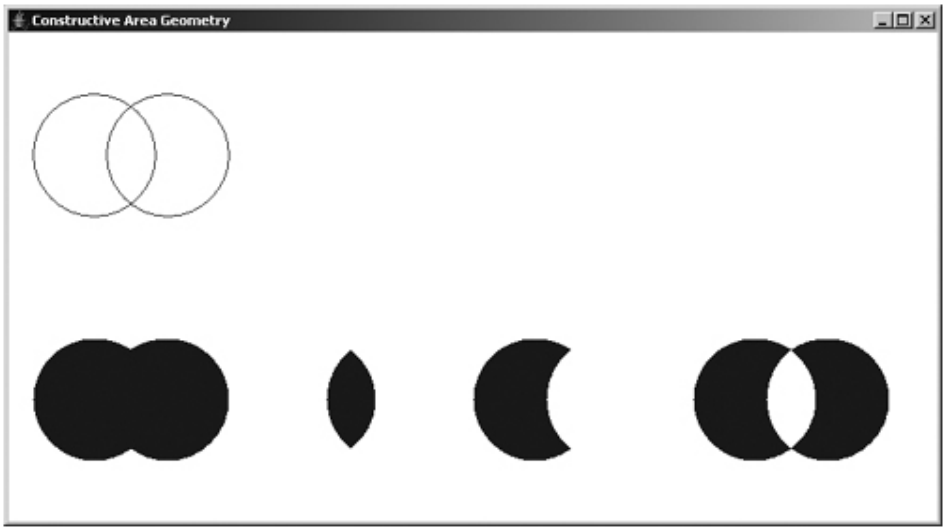
Gambar 2.7 Elips

2.9. Constructive Area Geometry

Salah satu cara untuk menciptakan bentuk yang lebih kompleks selain bentuk geometri dasar (garis, segi empat, lingkaran, dsb.) adalah dengan mengkombinasikan bentuk-bentuk dasar tersebut. Teknik seperti ini dinamakan *constructive area geometry*. Empat operasi yang didukung di antaranya: *union, intersection, difference*, dan *symmetric difference*. Operasi-operasi tersebut diterapkan pada dua daerah sehingga menghasilkan

sebuah daerah baru. *Union* dari dua daerah menghasilkan sebuah daerah baru yang merupakan hasil penggabungan kedua daerah asal. *Intersection* dari dua daerah menghasilkan sebuah daerah baru yang merupakan daerah irisan dari kedua daerah tersebut. *Difference* dari dua daerah menghasilkan sebuah daerah baru yang terdiri dari titik-titik yang terletak pada daerah pertama saja dan bukan daerah kedua. *Symmetric difference* dari dua daerah menghasilkan sebuah daerah baru yang titik-titik penyusunnya terdiri dari titik-titik yang hanya terletak pada tepat satu dari kedua daerah tersebut [2].

Gambar 2.8 menunjukkan contoh bentuk yang dihasilkan dari bentuk geometri lingkaran. Baris pertama pada gambar, menggambarkan dua bentuk lingkaran pertama. Baris di bawahnya menggambarkan hasil kombinasi menggunakan keempat operasi (*add*, *intersect*, *subtract*, *exclusiveOr*) menghasilkan empat bentuk baru.

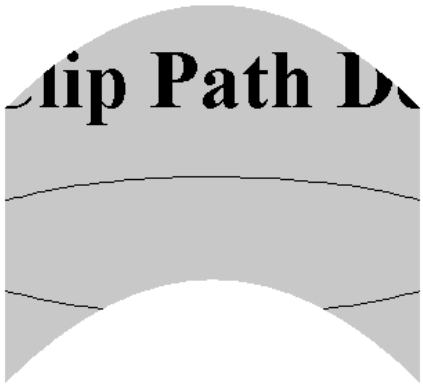


Gambar 2.8 Hasil gambar dengan teknik *constructive area geometry*

2.10. Clipping

Sebuah *clipping path* menentukan daerah mana saja dari objek yang akan terlihat pada piranti keluaran. Bagian objek yang berada di luar *clipping path* ini tidak akan digambar pada piranti keluaran. Berbagai macam bentuk dapat digunakan sebagai daerah *clipping* [2].

Gambar 2.9 mengilustrasikan contoh aplikasi sebuah *clipping path*.



Gambar 2.9 Contoh penggunaan *clipping path*

2.11. Teks & Font

Dalam komputer grafik, teks dipandang sebagai objek dengan bentuk geometri tertentu. Bentuk geometri yang menggambarkan bentuk karakter ini disebut dengan istilah *glyph*. Jenis *font* tertentu merupakan sekumpulan *glyph* untuk seluruh karakter alfabet.

2.12. Transformasi

Dengan menggunakan primitif-primitif untuk menggambar, kita dapat menghasilkan berbagai bentuk, gambar, warna dan pola yang diinginkan. Meskipun demikian, kadang kala dalam beberapa aplikasi muncul kebutuhan untuk memanipulasi bentuk/gambar yang telah dibuat sebelumnya.

Misalnya, perangkat lunak untuk menggambar seperti Corel Draw menyediakan berbagai macam tool untuk memindahkan, memperbesar, memperkecil, dan memutar gambar yang telah dibuat sebelumnya sehingga user dapat dengan mudah mengubah bentuk gambar tanpa harus menggambar ulang. Contoh lain misalnya, animasi komputer grafik untuk membuat film kartun atau animasi 3D membutuhkan transformasi untuk menimbulkan efek-efek pergerakan dan perubahan.

Manipulasi semacam ini dapat dilakukan dengan menerapkan transformasi terhadap suatu bentuk/gambar. Transformasi melakukan perubahan terhadap suatu bentuk asal menjadi bentuk baru sesuai dengan parameter-parameter yang telah ditentukan. Terdapat beberapa jenis transformasi, antara lain translasi (perpindahan), rotasi, dan penskalaan [1].

2.12.1 Translasi

Tranlasi merupakan jenis transformasi yang paling sederhana. Translasi berfungi untuk memindahkan objek dari suatu posisi ke posisi lain dengan pergerakan lurus. Kita dapat melakukan translasi terhadap sebuah titik yang berada di posisi (X,Y) dengan menambahkan masing-masing komponen koordinatnya dengan Jarak Translasi, Tx dan Ty, sehingga titik tersebut berpindah ke posisi baru (X', Y') :

$$X' = X + Tx$$

$$Y' = Y + Ty$$

Pasangan jarak translasi (Tx, Ty) disebut juga vektor translasi atau vektor perpindahan. Persamaan translasi di atas dapat juga dituliskan dalam bentuk matriks :

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad T = \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

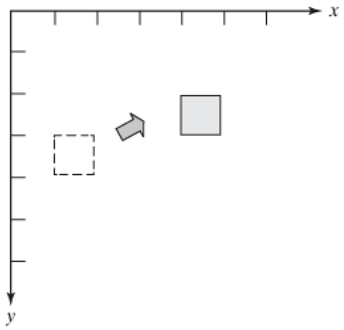
$$P' = P + T$$

Matriks transformasi untuk translasi adalah:

$$P' = M_T . P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Setiap gambar pada komputer selalu dapat direduksi menjadi sekumpulan titik, jadi benda yang lebih rumit dapat ditranslasi dengan memindahkan titik-titik penyusun gambarnya. Untuk mentranslasikan sebuah poligon, kita dapat menambahkan jarak translasi terhadap koordinat titik-titik sudutnya, kemudian menggambar ulang poligon tersebut pada titik-titik sudut yang telah ditranslasi.

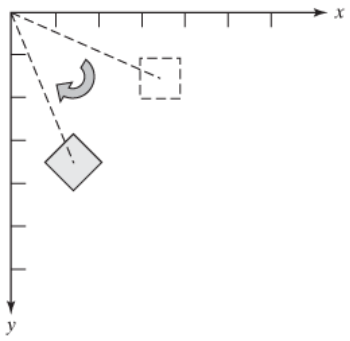


Gambar 2.10 Translasi (3,-1)

Jika yang ingin ditranslasikan adalah bentuk-bentuk seperti lingkaran atau elips, maka kita hanya perlu memindahkan titik pusatnya dan kemudian menggambar kembali lingkaran tersebut pada titik pusat hasil translasi. Begitu pula dengan bentuk kurva seperti bezier curve, untuk mentranslasikannya, hanya perlu dipindahkan titik-titik kontrol penyusunnya, kemudian menggambar ulang kurva tersebut dengan posisi titik kontrol yang baru.

2.12.2 Rotasi

Rotasi digunakan untuk memutar objek terhadap suatu titik putar tertentu. Untuk melakukan rotasi, kita perlu menentukan dua hal, yaitu sudut putar (θ) dan posisi titik putar (Xr, Yr) yang menjadi pusat perputaran objek:



Gambar 2.11 Rotasi dengan sudut putar (θ) dan posisi titik pusat putar (Xr, Yr)

Matriks transformasi rotasi sebesar Φ^0 dengan sudut pusat rotasi (0,0) adalah:

$$\begin{bmatrix} cos\ \emptyset & sin\ \emptyset & 0 \\ -sin\ \emptyset & cos\ \emptyset & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.12.3 Penskalaan

Penskalaan digunakan untuk mengubah ukuran objek. Operasi penskalaan dilakukan dengan cara mengalikan setiap koordinat penyusun objek (X,Y) dengan faktor skala (Sx, Sy) menghasilkan koordinat baru (X',Y').

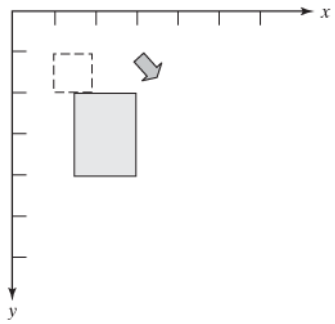
$$X' = X \cdot Sx$$
$$Y' = Y \cdot Sy$$

Dalam bentuk matriks dapat dituliskan menjadi:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Faktor skala Sx mengatur ukuran objek dalam arah sumbu X, dan Sy dalam arah sumbu Y. Setiap nilai numerik positif dapat digunakan sebagai faktor skala. Nilai yang lebih kecil dari 1 bersifat memperkecil ukuran objek, sedangkan nilai yang lebih besar dari 1 akan memperbesar objek.

Jika nilai Sx dan Sy sama, maka didapatkan perubahan ukuran objek yang proporsional dalam arah X dan Y (uniform scaling). Jika Sx dan Sy berbeda (differential scaling), maka perubahan ukuran objek menjadi berbeda dalam arah X dan Y, sehingga objek bisa menjadi lebih lebar atau lebih tinggi.



Gambar 2.12 Skala dengan faktor (1.5, 2)

Operasi penskalaan tidak hanya mengubah ukuran objek, tetapi juga mengubah posisi objek. Posisi objek berubah karena operasi penskalaan menggunakan rumus di atas sebenarnya adalah penskalaan relatif terhadap titik pusat koordinat (0,0). Jadi jika sebuah objek diperbesar, maka posisi objek bergeser menjauh dari pusat koordinat, jika diperkecil maka bergerak mendekati pusat koordinat.

Agar operasi penskalaan tidak mengubah posisi objek, maka perlu ditentukan suatu posisi titik pusat penskalaan (*fixed point*). *Fixed point* (X_f,Y_f) tersebut pertama-tama dipindahkan ke titik pusat koordinat (0,0) dengan translasi (-X_f,-Y_f). Setelah *fixed point* berpindah ke pusat koordinat, baru skalakan setiap titik yang telah ditranslasi tadi dengan faktor skala (S_x, S_y). Setelah diskalakan, kembalikan semua titik tadi ke posisi asalnya dengan translasi (X_f,Y_f). Secara ringkas dapat dituliskan dengan persamaan :

$$x' = x_f + (x - x_f).S_x$$

$$y' = y_f + (y - y_f).S_y$$

Persamaan ini dapat ditulis ulang menjadi :

$$x' = x.S_x + (1 - S_x).x_f$$

$$y' = y.S_y + (1 - S_y).y_f$$

Dengan persamaan ini, nilai $(1\text{-}S_x).X_f$ dan $(1\text{-}S_y).Y_f$ hanya perlu dihitung satu kali untuk setiap titik yang akan ditransformasi sehingga dapat mempercepat perhitungan.

2.12.4 Komposisi Transformasi

Transformasi dapat berupa kombinasi dari beberapa jenis transformasi. Sebagai contoh, kombinasi translasi, rotasi, diakhiri dengan translasi kembali. Jenis transformasi seperti ini dikenal dengan istilah komposisi transformasi. Jenis transformasi ini membutuhkan matriks transformasi dalam prosesnya. Matriks transformasi ini adalah hasil *dot product* dari beberapa matriks transformasi tunggal. Sebagai contoh, jika M_1, M_2, M_3 masing-masing adalah matriks transformasi tunggal (T_1, T_2, T_3), maka matriks komposisi dari ketiga transformasi T_1 º T_2 º T_3 adalah $M_1.M_2.M_3$.

Yang perlu ditekankan adalah komposisi transformasi tidak bersifat komutatif, artinya urutan sangat penting. Perbedaan urutan transformasi menghasilkan hasil yang berbeda. Pada pembahasan ini, komposisi transformasi diaplikasikan dari kanan ke kiri. Sebagai contoh, jika komposisi transformasi T_1 º T_2 º T_3 diaplikasikan pada sebuah titik p, urutan proses transformasi yang dilakukan adalah T_3, T_2, T_1:

$(T_1 \text{ º } T_2 \text{ º } T_3)(p) = T_1(T_2(T_3(p)))$

Komposisi transformasi sangat bermanfaat untuk membangun transformasi kompleks yang disusun dari transformasi-transformasi lain yang lebih sederhana. Jika ingin dilakukan rotasi 300 dengan titik (3,4) sebagai titik pusat rotasi, yang perlu dilakukan adalah:

1. Pertama, memindahkan titik (3,4) ke titik pusat koordinat (0,0).

2. Kemudian lakukan rotasi 300 terhadapat titik pusat (0,0).

3. Dan terakhir, translasikan hasil rotasi tersebut kembali ke posisi asalnya di (3,4).

Dengan mengkombinasikan ketiga transformasi tersebut, akan didapatkan hasil rotasi seperti yang diharapkan.

1. Dalam bentuk matriks, proses translasi untuk memindahkan titik (3,4) ke titik pusat adalah:

$$\begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Matriks rotasi 30^0 terhadap titik pusat adalah:

$$\begin{bmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Matriks translasi kedua adalah:

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Dengan mengkombinasikan ketiga transformasi tersebut, matriks transformasi yang terbentuk adalah:

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

2.13. Animasi Sederhana

Animasi atau gambar bergerak merupakan salah satu aplikasi grafika komputer yang paling banyak digunakan di dunia hiburan, pendidikan, penelitian, dan industri. Banyak teknik yang dapat digunakan untuk menciptakan animasi, dari yang paling sederhana sampai yang rumit sekali. Hari ini, kita akan mencoba membuat animasi sederhana.

Prinsip dasar animasi adalah mengubah gambar secara cepat dan menampilkan gambar yang sedikit berbeda dengan gambar sebelumnya sehingga timbul ilusi gerakan pada mata yang melihatnya. Animasi manual dapat dibuat dengan menggambar objek-objek dengan perubahan bertahap pada beberapa lembar kertas kosong, kemudian kita lihat hasilnya dengan membalikkan kertas-kertas tersebut dengan cepat.

Untuk membuat animasi di layar komputer, prinsip yang sama tetap berlaku. Serupa dengan membalikkan kertas, sekarang kita lakukan serangkaian penggambaran dan penghapusan secara bergantian pada monitor.

Ide sederhananya :

Gambar bentuk awal
Hapus
Gambar bentuk kedua
Hapus
Gambar bentuk ketiga
Hapus
...
dst sampai berulang-ulang

Pada implementasi sebenarnya, setelah penggambaran objek biasanya ada waktu tunggu (*delay*) sehingga animasi tidak berlangsung terlalu cepat. Besarnya waktu tunggu ini akan berpengaruh pada kecepatan animasi [1].

2.14. Image Processing

Sebuah citra digital merupakan susunan *pixel-pixel* dengan warna tertentu. Masing-masing nilai *pixel* tersebut mendefinisikan warna, *graylevel*, dan nilai atribut lain yang berhubungan dengan informasi *pixel*. Konsep dasar pengolahan citra (*image processing*) sebenarnya cukup sederhana. Proses pengolahan citra adalah masalah untuk menghitung warna baru untuk setiap *pixel* pada gambar. Warna *pixel* baru dapat diperoleh berdasarkan warna *pixel* sebelumnya, warna *pixel-pixel* tetangganya, parameter tertentu, atau kombinasi dari ketiganya [2].

Beberapa efek yang dapat diciptakan melalui proses pengolahan citra di antaranya *blur, sharpen, grayscale,* dan sebagainya.

Berikut adalah contoh gambar hasil pengolahan citra (*image processing*). Gambar paling kiri adalah citra asal, yang kemudian dengan proses Blur menghasilkan citra tengah, dan dengan proses Sharpen menghasilkan citra paling kanan.

Citra asal Blur Sharpen

Gambar 2.13 Hasil proses pengolahan citra (*image processing*)

2.14.1 Blurring

Efek *blur* adalah efek buram seperti hasil foto yang tidak fokus. Hal yang sebenarnya terjadi pada citra adalah *pixel-pixel* penyusun gambar asal disebar dan dicampur dengan *pixel-pixel* tetangganya. Untuk melakukan hal ini, dilakukan proses konvolusi. Proses konvolusi adalah proses menjalankan sebuah matriks/kernel konvolusi pada setiap *pixel* gambar asal. Nilai pada matriks konvolusi kemudian dikalikan dengan nilai *pixel* yang bersesuaian di gambar asal, kemudian nilai-nilai tersebut ditambahkan untuk menjadi nilai *pixel* baru dengan posisi yang bersesuaian di gambar tujuan.

Sebagai contoh, efek *blur* yang paling sederhana dapat diperoleh dengan merata-ratakan nilai suatu *pixel* dengan delapan *pixel* tetangganya. Kernel konvolusi untuk melakukan hal ini adalah:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Jumlah seluruh elemen matriks ini adalah 1, yang berarti tingkat pencahayaan gambar asal akan sama dengan tingkat pencahayaan gambar hasil.

Gambar 2.14 memperlihatkan contoh hasil gambar yang diberi efek *blur.*

Gambar 2.14 Gambar asal dan gambar hasil efek *blur*

2.14.2 Sharpen

Berlawanan dengan efek *blur,* efek *sharpen* adalah efek mempertajam gambar. Efek ini dapat dihasilkan dengan kernel konvolusi:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Gambar 2.15 adalah contoh gambar dengan efek *sharpen.*

Gambar 2.15 Gambar yang diberi efek *sharpen*

2.14.3 Edge Detection

Efek *edge detection* merupakan salah satu efek yang banyak digunakan dalam pemrosesan gambar, *machine vision, computer vision, feature detection*, dan *feature extraction.* Seperti namanya, efek ini menghasilkan garis-garis tepian gambar seperti sketsa gambar. Cara kerjanya adalah dengan mengidentifikasi perubahan tingkat gelap terangnya *pixel* pada gambar. Gambar 2.16 merupakan contoh gambar asal dan contoh gambar hasil dengan efek *edge detection.* Efek ini dapat dihasilkan dengan kernel konvolusi:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

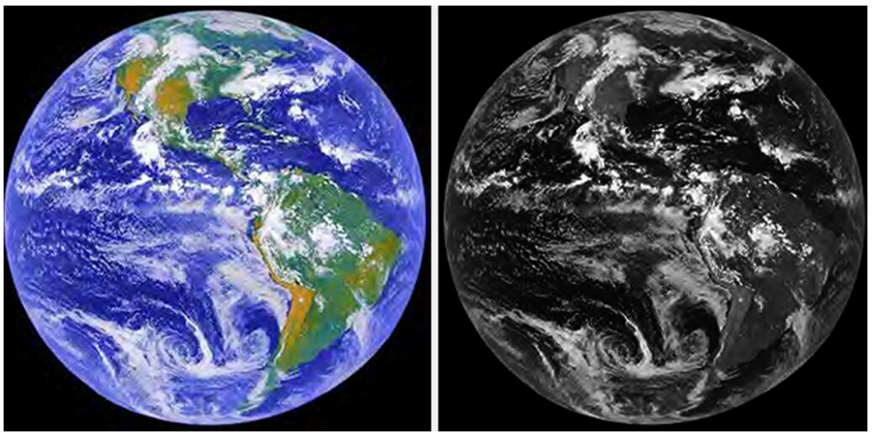
Gambar 2.16 Efek *edge detection* pada gambar

2.14.4 Grayscale

Efek *grayscale* dapat dihasilkan dengan meng-*assign* ketiga komponen warna RGB (*Red, Green, Blue*) setiap *pixel* pada gambar hasil dengan rata-rata ketiga nilai tersebut pada gambar asal.

$$R' = G' = B' = \frac{R + G + B}{3}$$

Gambar 2.17 merupakan contoh gambar dengan efek *grayscale.*



Gambar 2.17 Gambar dengan efek grayscale

BAB III
RENDERING ENGINE JAVA2D

3.1. Java2D

Sebenarnya, hampir semua bahasa pemrograman modern memiliki dukungan untuk menggambar di layar, antara lain Java, C/C++, Pascal, Visual Basic, LISP, bahkan Prolog. Pada kesempatan ini, akan digunakan bahasa pemrograman Java untuk melakukan eksperimen grafika 2D.

Pertanyaannya: “Mengapa Java?” Bahasa pemrograman Java sudah semakin banyak digunakan baik dalam aplikasi maupun dalam materi pembelajaran. Hal ini menyebabkan *library* tambahan Java, seperti Java2D/3D, cocok digunakan untuk mempelajari grafika komputer. Selain itu, aplikasi multimedia Java banyak digunakan sebagai bahasa pengembang pilihan pertama.

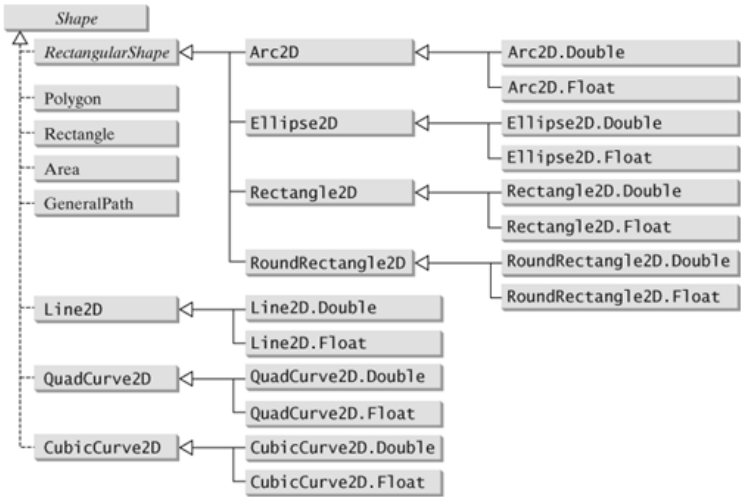
3.2. Model Geometri

Library Java2D telah menyediakan *method-method* untuk menggambar bentuk-bentuk geometri standar seperti garis, lingkaran, elips, segiempat, dsb. Sebuah objek geometri dapat di-*render* dengan mengimplementasikan *interface* Shape. Graphics2D memiliki *method* draw(Shape s) dan fill(Shape s) untuk menggambar bentuk *outline* dan bentuk penuh sebuah objek geometri. Java 2D dapat membentuk bentuk-bentuk geometri dasar, kemudian menggabungkannya untuk membentuk objek lain yang lebih kompleks [2]. Hierarki *class* Shape dapat dilihat pada gambar 3.1.

Class Line2D, QuadCurve2D, CubicCurve2D, Rectangle2D, RoundRectangle2D, Arc2D, dan Ellipse2D merupakan abstract class. Masing-masing memiliki dua inner subclasses: X.Double dan X.Float, yang masing-masing menyatakan nilai koordinatnya menggunakan tipe data double atau float. Sebagai contoh, Line2D.Double dan Line2D.Float adalah 2 sub-classes dari Line2D. Kedua class tersebut sama-sama menggambar garis, tetapi berbeda dalam representasi nilai koordinatnya. Untuk membuat

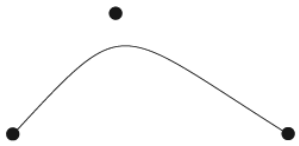
sebuah objek Line2D dengan tipe double, gunakan *constructor*:

```
Line2D line = new Line2D.Double(x1, y1, x2, y2);
```



Gambar 3.1 Hierarki Class Shape

QuadCurve2D merepresentasikan sebuah quadratic curve yang didefinisikan dengan tiga buah titik kontrol. Titik pertama dan terakhir merupakan titik-titik ujung kurva. Titik kedua biasanya tidak terletak pada kurva, tetapi menunjukkan kelengkungan kurva, seperti ditunjukkan pada gambar 3.2.



Gambar 3.2 QuadCurve2D didefinisikan dengan 3 titik kontrol

Sebuah objek QuadCurve2D dapat diciptakan dengan *constructor*:

```
QuadCurve2D quad = new QuadCurve2D.Double(x1,y1,x2,y2,x3,y3);
```

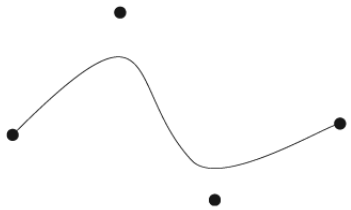
CubicCurve2D merepresentasikan kurva *cubic Bezier* yang didefinisikan dengan 4 titik kontrol. Sama seperti *quadratic curve*, titik kontrol pertama dan terakhir adalah titik ujung kurva. Dua titik kontrol di antaranya mendefinisikan bentuk kelengkungan kurva, dan tidak terletak pada kurva, seperti ditunjukkan pada gambar 3.3.

Rectangle2D mendefinisikan bentuk segi empat, didefinisikan dengan sebuah titik ujung pertama dan panjang dan lebar dari bentuk segi empat tersebut. Bentuk ini diciptakan dengan *constructor*:

```
Rectangle2D ri = new Rectangle(x0,y0,length,width);
```

```
Rectangle2D rd = new
Rectangle2D.Double(x0.0,y0.0,length.0,width.0);

Rectangle2D rf = new
Rectangle2D.Float((x0)f,(y0)f,(length)f,(width)f);
```



Gambar 3.3 CubicCurve2D didefinisikan dengan 4 titik kontrol

Ketiganya sama-sama membentuk segi empat, perbedaannya hanya terletak pada tipe data koordinatnya. Yang pertama bertipe integer, yang kedua dan ketiga masing-masing bertipe double dan float. Parameter pertama dan kedua mendefinisikan koordinat titik ujung pertama. Parameter ketiga dan keempat mendefinisikan panjang dan lebar dari segi empat tersebut.

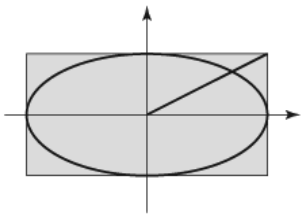
RoundRectangle2D mendefinisikan bentuk segi empat dengan bentuk ujung membulat. Dua parameter tambahan dari Rectangle2D menspesifikasikan lebar dan tinggi bentuk membulat tersebut. Sebagai contoh, *constructor* berikut menciptakan bentuk *round rectangle* dengan dimensi 5x5:

```
RoundRectangle2D rrect = new
RoundRectangle2D.Double(20,30,100,80,5,5);
```

Ellipse2D mendefinisikan bentuk elips penuh dengan *constructor*:

```
Ellipse2D ellipse = new
Ellipse2D.Float((xup)f,(yup)f,(xbr)f,(ybr)f);
```

Empat parameter dari *constructor* tersebut mendefinisikan koordinat titik kiri atas (up=*upper left*) dan kanan bawah (br=*bottom right*) dari *boundary box* elips tersebut.



Gambar 3.4 Elips dengan *boundary box*-nya

Arc2D mendefinisikan bentuk busur elips dengan *constructor*:

```
Arc2D arc = new
Arc2D.Float((xup)f,(yup)f,(xbr)f,(ybr)f,(θ1)f,(θ2)f,Arc2D.P
IE);
```

Sama seperti elips, empat parameter pertama dari *constructor* tersebut mendefinisikan koordinat titik kiri atas (up=*upper left*) dan kanan bawah (br=*bottom right*) dari *boundary box* elips tersebut. Dua parameter berikutnya menyatakan *range* sudut yang dibentuk oleh busur (dalam derajat). Parameter terakhir menunjukkan tiga pilihan penutupan busur: OPEN, CHORD, atau PIE.

Class Polygon hanya dapat mendefinisikan koordinat titik dengan tipe data integer. Poligon didefinisikan dengan *constructor*:

```
Polygon(int[] xcoords, int[] ycoords, int npoints);
```

Kedua array integer mendefinisikan vertex pembentuk polygon. Titik pertama dan titik terakhir kemudian dihubungkan membentuk kurva tertutup.

3.3. Constructive Area Geometry

Salah satu cara untuk menciptakan bentuk yang lebih kompleks selain bentuk geometri dasar (segi empat, lingkaran, dsb.) adalah dengan mengkombinasikan bentuk-bentuk dasar tersebut. Teknik seperti ini dinamakan *constructive area geometry*. Empat operasi yang didukung di antaranya: *union*, *intersection*, *difference*, dan *symmetric difference*. Operasi-operasi tersebut diterapkan pada dua daerah sehingga menghasilkan sebuah daerah baru. *Union* dari dua daerah menghasilkan sebuah daerah baru yang merupakan hasil penggabungan kedua daerah asal. *Intersection* dari dua daerah menghasilkan sebuah daerah baru yang merupakan daerah irisan dari kedua daerah tersebut. *Difference* dari dua daerah menghasilkan sebuah daerah baru yang terdiri dari titik-titik yang terletak pada daerah pertama saja dan bukan daerah kedua. *Symmetric difference* dari dua daerah menghasilkan sebuah daerah baru yang titik-titik penyusunnya terdiri dari titik-titik yang hanya terletak pada tepat satu dari kedua daerah tersebut.

Daerah objek ini dibentuk menggunakan *constructor*:

```
Area(Shape s)
```

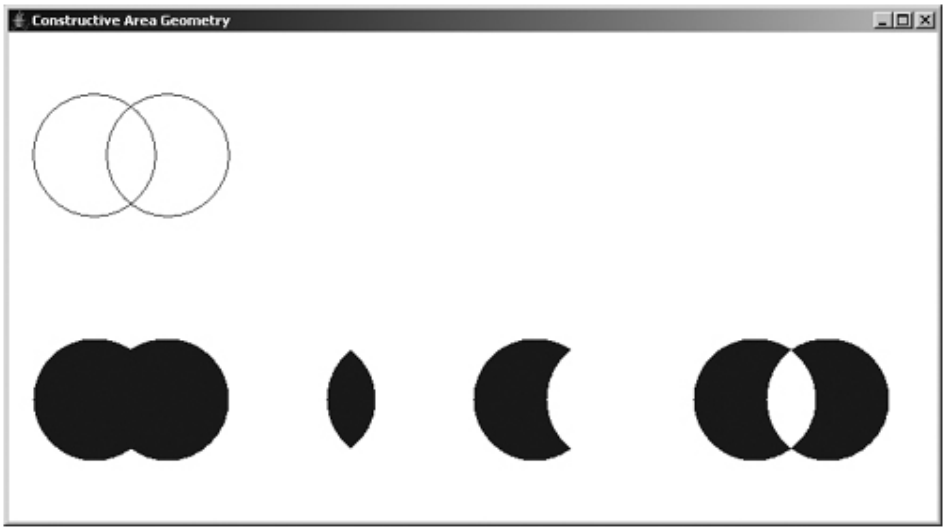
Empat operasi yang dapat dilakukan dengan *method-method* dari sebuah objek `Area`:

```
void add(Area a)
void intersect(Area a)
```

```
void subtract(Area a)
void exclusiveOr(Area a)
```

Hasil dari operasi-operasi tersebut disimpan pada objek `Area` yang aktif saat ini. Objek `Area` yang dikirimkan pada parameter *method* tidak akan berubah akibat pemanggilan operasi tersebut.

Baris pertama menggambarkan dua bentuk pertama. Baris di bawahnya menggambarkan hasil kombinasi menggunakan keempat operasi (*add, intersect, subtract, exclusiveOr*) menghasilkan empat bentuk baru.



Gambar 3.5 Hasil gambar dengan teknik constructive area geometry

Pada eksperimen tersebut, dua buah lingkaran yang saling berpotongan, sebutlah s_1 dan s_2, digunakan sebagai bentuk asal. Kedua bentuk tersebut digambarkan pada baris pertama gambar. Dari kedua bentuk tersebut, dibentuklah objek Area a_1 dan a_2 (baris 33-34). Empat operasi (*add, intersect, subtract, exclusiveOr*) kemudian diterapkan pada kedua objek `Area`. Hasilnya digambarkan pada baris bawah gambar (baris 40-53). Karena *class* `Area` mengimplementasikan *interface* `Shape`, objek `Area` dapat secara langsung dikirimkan sebagai parameter *method* `fill` dari objek `Graphics2D`.

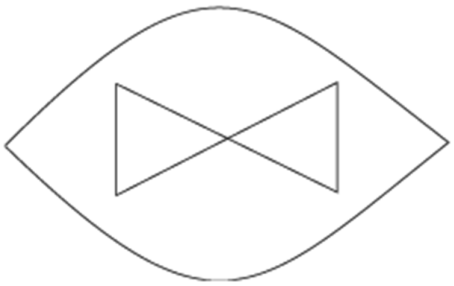
3.4. General Path

Proses penggambaran pada Java2D dapat dilakukan seperti menggambar dengan pena. *Class* yang digunakan adalah class `GeneralPath`. Pada suatu waktu tertentu, pena akan berada pada posisi tertentu. *Method* `moveTo(float x,float y)` memindahkan posisi pena ke lokasi baru (x,y) tanpa menggambar apapun. *Method* `lineTo(float`

`x,float y)` menggambar garis dari posisi pena saat ini ke titik (x,y) kemudian men-set posisi akhir pena di titik tersebut. *Method* `quadTo(float x1,float y1,float x2,float y2)` menggambar kurva dari posisi pena saat ini ke titik (x2,y2) menggunakan titik (x1,y1) sebagai titik kontrol tengah. *Method* `curveTo(float x1,float y1,float x2,float y2,float x3,float y3)` menggambar kurva dari posisi pena saat ini ke titik (x3,y3) menggunakan titik (x1,y1) dan (x2,y2) sebagai dua titik kontrol tengah. *Method* `closePath` membentuk kurva tertutup dengan menggambar garis ke titik yang menjadi parameter pemanggilan *method* `moveTo` terakhir.

Sebagai contoh, potongan program berikut menggambar bentuk seperti ditunjukkan pada gambar di bawahnya.

```
GeneralPath path = new GeneralPath();
path.moveTo(-2f, 0f);
path.quadTo(0f, 2f, 2f, 0f);
path.quadTo(0f, -2f, -2f, 0f);
path.moveTo(-1f, 0.5f);
path.lineTo(-1f, -0.5f);
path.lineTo(1f, 0.5f);
path.lineTo(1f, -0.5f);
path.closePath();
```



Gambar 3.6 Contoh bentuk menggunakan `GeneralPath`

3.5. Clipping

Sebuah clipping path menentukan daerah mana saja dari objek yang akan terlihat pada piranti keluaran. Bagian objek yang berada di luar clipping path ini tidak akan digambar pada piranti keluaran. Berbagai macam bentuk dapat digunakan sebagai daerah clipping.

Potongan program di bawah ini menggunakan sebuah elips sebagai bentuk clipping sebuah gambar. Artinya, hanya gambar yang berada di dalam bentuk elips ini saja yang akan terlihat pada piranti keluaran.

```
Graphics2D g2 = (Graphics2D)g;
Shape ellipse = new Ellipse2D.Double(0, 0, 300,200);
g2.setClip(ellipse);
g2.drawImage(image, 0, 0, this);
```

Method lain dari Graphics2D yang dapat digunakan untuk mengubah daerah *clip* adalah: `void clip(Shape path)`

Method ini akan memotong daerah *clipping* saat ini dengan bentuk yang diberikan pada parameter.

Gambar 3.7 merupakan contoh hasil *clipping* sebuah gambar langit dengan teks JAVA sebagai daerah batas *clipping* (pemotongannya).

Gambar 3.7 Contoh *clipping* sebuah gambar dengan teks JAVA

3.6. Teks & Font

Java 2D menyediakan banyak sekali fitur manipulasi *font* dan teks. Method yang paling sering digunakan untuk manipulasi karakter adalah `setFont` dan `drawString` pada `Graphics2D`.

Sebuah objek `Font` diciptakan melalui pemanggilan constructor:

```
Font(String name, int style, int size)
```

Parameter `name` menspesifikasikan nama font yang ingin digunakan, misal: “Times New Roman”. Font yang tersedia pada setiap setiap tentu berbeda-beda. Karenanya, Java menyediakan lima logical font yang memetakan kelima pilihan logical font tersebut ke font pada sistem tertentu. Sebagai contoh, logical font “SansSerif” dipetakan pada “Arial” pada sistem Windows. Lima logical font yang didukung oleh Java adalah:

```
Serif
SansSerif
Monospaced
Dialog
DialogInput
```

Parameter `style` menspesifikasikan gaya font:

`PLAIN`

`ITALIC`

`BOLD`

Ketiganya dapat dikombinasikan dengan operator bitwise OR "|".

Parameter `size` menspesifikasikan ukuran *font*.

Objek Font dapat dipilih dari objek Graphics2D dengan method:

```
void setFont(Font font)
```

Font ini kemudian akan menjadi jenis *font* aktif yang digunakan pada setiap pemanggilan *method* penggambaran teks selanjutnya:

```
void drawString(String s,int x,int y)
void drawString(String s,float x,float y)
```

Method `getStringBounds` me-*return bounding rectangle* dari string.

3.7. Warna

Pada Graphics2D, sebuah bentuk geometri digambar menggunakan *method* `fill(Shape)` atau `draw(Shape)`. Untuk mengatur warna untuk bentuk geometri, digunakan *method*:

```
void setColor(Color c)
```

Objek `Color` mendefinisikan warna objek, terdiri dari 4 komponen: komponen Red, Green, Blue, (RGB) dan Alpha. Komponen Alpha mendefinisikan transparansi warna. Nilai konstan warna di antaranya: `black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow`.

Selain warna-warna tersebut, berbagai kombinasi nilai RGB dan Alpha dapat dihasilkan melalui *constructor*:

```
Color(int r, int g, int b);
Color(int rgb);
Color(float r, float g, float b);
```

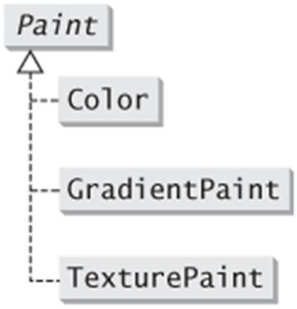
Constructor pertama menerima tiga nilai int dengan range 0-255. *Constructor* kedua membungkus ketiga nilai tersebut dalam sebuah nilai int. *Constructor* ketiga menerima nilai *float* dengan range 0.0 - 1.0.

Jika komponen Alpha (untuk transparansi warna) ingin diperhitungkan, gunakan *constructor*:

```
Color(int r, int g, int b, int a);
Color(int rgba, boolean hasAlpha);
Color(float r, float g, float b, float a);
```

3.8. Paint

Method `void setColor(Color c)` mengatur warna solid dalam proses penggambaran objek geometri. Jika warna yang diinginkan untuk menggambar merupakan warna yang bergradasi, sebuah *class* `GradientPaint` dapat mendefinisikan kombinasi warna melalui definisi 2 titik dan 2 warna. Warna pertama akan berubah perlahan-lahan menjadi warna kedua dari lokasinya di titik pertama ke lokasinya di titik kedua. Bentuknya bisa dibuat melingkar atau tidak. Pola seperti ini akan dibuat berulang.



Gambar 3.8 Hierarki *class* Paint

Untuk membuat warna yang tidak melingkar, gunakan *constructor*:

```
GradientPaint(float x1,float y1,Color c1,float x2,float y2,
Color c2);
GradientPaint(Point2D p1,Color c1,Point2D p2,Color c2);
```

Untuk membuat warna dengan gradasi berbentuk melingkar, gunakan *constructor*:

```
GradientPaint(float x1,float y1,Color c1,float x2,float
y2,Color c2,boolean cycl);
GradientPaint(Point2D p1, Color c1, Point2D p2, Color c2,
boolean cycl);
```

Kemudian set parameter terakhir dengan nilai true.

Class TexturePaint mengisi bentuk sebuah objek dengan tekstur tertentu. Sebuah gambar biasanya digunakan untuk mendefinisikan tekstur ini. Objek ini diciptakan dengan *constructor*:

```
TexturePaint(BufferImage image, Rectangle2D anchor);
```

`Image` digunakan untuk mendefinisikan tekstur yang akan digunakan untuk menggambar, anchor mendefinisikan posisi gambar pada hasil keluaran.

3.9. Transformasi: Translasi

Dalam Java, translasi dilakukan melalui perintah:

```
AffineTransform t = new AffineTransform();
t.setToTranslation(double tx, double ty);
```

Nilai tx dan ty pada parameter *method* setToTranslation di atas adalah nilai perpindahan dalam arah x dan perpindahan dalam arah y.

3.10. Transformasi: Rotasi

Dalam Java, rotasi dilakukan melalui perintah:

```
AffineTransform t = new AffineTransform();
t. setToRotation(double theta);
```

Nilai theta pada parameter setToRotation menyatakan besarnya sudut rotasi.

Dalam Java, rotasi dapat pula dilakukan melalui perintah:

```
AffineTransform t = new AffineTransform();
t.setToRotation(double theta,double x,double y);
```

Nilai `theta` pada parameter `setToRotation` menyatakan besarnya sudut rotasi. Nilai `x` dan `y` nya menyatakan titik pusat rotasi.

3.11. Transformasi: Penskalaan

Dalam Java, scaling dilakukan melalui perintah:

```
AffineTransform scaling = new AffineTransform();
scaling.setToScale(double sx, double sy);
```

Nilai `sx` dan `sy` pada parameter `setToScale` menyatakan besarnya penskalaan dalam arah x dan dalam arah y.

3.12. Transformasi: Komposisi Transformasi

Dalam Java, komposisi transformasi didukung oleh method-method:

```
void rotate(double theta)
void scale(double sx, double sy)
void shear(double shx, double shy)
void translate(double tx, double ty)
```

Untuk mengkombinasikan transformasi-transformasi tunggal tersebut, digunakan *method*:

```
void concatenate(AffineTransform tx)
void preConcatenate(AffineTransform tx)
```

Method pertama menggabungkan transformasi lain ke sebelah kanan transformasi saat ini. *Method* kedua menggabungkannya di sebelah kiri.

Jika suatu komposisi transformasi dibuat dengan *method-method* di atas, transformasi diaplikasikan dalam urutan yang **berlawanan** dengan urutan pemanggilan *method*.

Pada potongan kode berikut, transformasi yang pertama diaplikasikan adalah translasi dan transformasi yang terakhir adalah rotasi.

```
AffineTransform transform = new AffineTransform();
transform.rotate(Math.PI/3);
transform.scale(2, 0.3);
transform.translate(100, 200);
```

3.13. Animasi Sederhana

Dalam Java, implementasi animasi membutuhkan *thread* tambahan untuk menangani perubahan gambar secara real-time. Cara yang ideal untuk membuat animasi pada Java adalah dengan memisahkan bagian rendering gambar dengan bagian untuk menampilkan frame (transisi perubahan gambarnya). Ketika data untuk masing-masing frame siap, lakukan pemanggilan *method* repaint. Garis besar pendefinisian animasi dalam Java adalah sbb.:

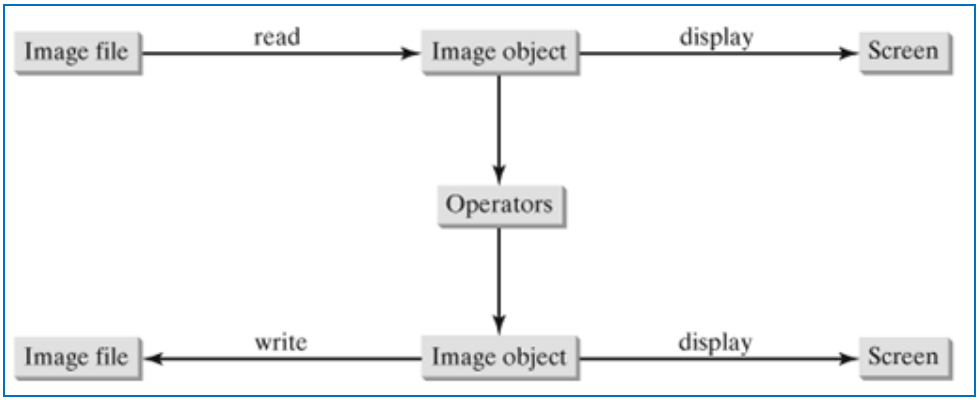
```
public void paintComponent(Graphics g) {
     <* render frame *>
}

public void run() {
     while(true) {
          <* update frame data *>
          repaint();
          try {
               Thread.sleep(sleepTime);
          }  catch  (InterruptedException  ex)
        {}
     }
```

Method `paintComponent` berisi kode untuk me-render sebuah frame. *Method* `run` dari *interface* `Runnable` dan *class* `Thread` di-overload untuk membuat animasi. Sebuah *frame* di-*render* dengan memanggil *method* `repaint`. Di antara dua buah *frame*, *thread* pada umumnya *sleep* selama beberapa saat (satuan milisecond). Hal ini dilakukan agar animasi yang dihasilkan menjadi tidak terlalu cepat.

3.14. Image Processing

Proses pengolahan citra pada Java 2D diperlihatkan pada gambar 3.9.



Gambar 3.9 Proses pengolahan citra pada Java2D

Sebuah objek `Image` dibentuk dalam program Java untuk merepresentasikan data file gambar asal. Format file gambar asal (*source image*) dapat bermacam-macam. Objek `Image` ini kemudian dapat ditampilkan pada perangkat keluaran seperti layar atau printer atau diproses dengan satu atau lebih operator pengolahan citra. Hasil operasi

pengolahan citra ini berupa objek `Image` lain yang dapat ditulis ke eksternal *file image* atau ditampilkan ke piranti keluaran.

Java 2D menggunakan `BufferedImage` untuk merepresentasikan gambar. Perintah Java untuk menciptakan objek ini:

```
BufferedImage bi = new BufferedImage
                 (int width,int height,int imageType);
```

Contoh pemanggilan:

```
BufferedImage bi = new BufferedImage
                 (300,400,BufferedImage.TYPE_INT_RGB);
```

Untuk menggambar pada sebuah BufferedImage, dibutuhkan bantuan objek Graphics2D:

```
Graphics2D g2 = (Graphics2D)(bi.createGraphics());
```

Dengan objek `Graphics2D` g2, semua jenis *graphics rendering* dapat digambar pada BufferedImage tersebut.

Terdapat dua cara untuk me-*load file* gambar:

1. Metode pertama menggunakan `static method read` dari *class* `ImageIOPanel` untuk me-*load file* gambar secara langsung ke objek `BufferedImage`.

   ```
   BufferedImage bi = ImageIO.read(file);
   ```

2. Metode kedua menggunakan AWT image loading:

   ```
   Image image=
   Toolkit.getDefaultToolkit().getImage(imageFileName);
   ```

   Dalam sebuah *applet*, sebuah gambar dapat di-*load* dari URL menggunakan *method* `getImage(url)` dari *class* `Applet`.

Objek `MediaTracker` digunakan sebagai waktu tunggu sampai sebuah gambar selesai di-*load*.

```
MediaTracker tracker = new MediaTracker(new Component() {});
tracker.addImage(image, 0);
try {
   tracker.waitForID(0);
} catch (InterruptedException ex) {
}
```

Constructor `MediaTracker` menerima sebuah parameter dari *class* `Component`. Karena `Component` adalah sebuah *abstract class*, tidak dapat diciptakan sebuah *generic instance* dari *class* tersebut. Oleh karena itu, pada potongan kode program di atas, digunakan sebuah sebuah *subclass* `new Component() {}` sebagai parameter.

Prosedur di atas membaca gambar ke dalam sebuah *object* `Image` dengan nama `image`. Untuk mengkonversi *object* `Image` tersebut menjadi sebuah *object* `BufferedImage`, digunakan *object* `g2` yang dihubungkan dengan *object* `BufferedImage`, `bi`. *Method* `drawImage` dari *object* `g2` digunakan untuk menggambar *image* ke `bi`.

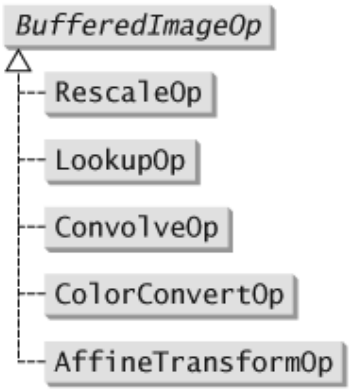
```
g2.drawImage(image, 0, 0, new Component() {});
```

Ukuran `bi` sama dengan ukuran *image*. Pemanggilan *method* di atas meng-konversi *object* `Image` tersebut menjadi sebuah *object* `BufferedImage`. Parameter terakhir dari *method* `drawImage` adalah `ImageObserver`. Karena *class* `Componet` mengimplementasikan *interface* `ImageObserver`, sebuah *subclass* baru tanpa nama dapat digunakan sebagai sebuah *generic parameter* atau nilai *null* sebagai nilai untuk `ImageObserver`. Jika kode ini berada pada sebuah aplikasi GUI, maka komponen GUI seperti *object* `JPanel` dapat digunakan sebagai *image observer*. *Object* `ImageObserver` digunakan untuk menggambar kembali komponen sehingga sebuah gambar dapat secara berulang-ulang digambar kembali.

Setelah sebuah *object* `BufferedImage` diciptakan, proses pengolahan citra (*image processing*) dapat dilakukan.

Gambar 3.10 Proses pengolah citra

Java 2D memiliki beberapa *class* untuk memproses gambar. *Class-class* ini mengimplementasikan *interface* `BufferedImageOp` seperti ditunjukkan pada gambar berikut:



Gambar 3.11 Class dalam Java2D untuk memproses gambar

RescaleOp

men-skalakan nilai *pixel* dengan sebuah fungsi linear. Penskalaan ini dilakukan untuk setiap *pixel* pada gambar. Nilai *pixel* dikalikan dengan sebuah faktor skala kemudian ditambahkan dengan sebuah nilai offset. Jika f(x,y) dan g(x,y) merepresentasikan sebuah nilai *pixel* pada sebuah gambar sebelum dan sesudah pemrosesan, operasi penskalaan ini dapat ditulis dengan persamaan:

$$g(x,y) = a.f(x,y) + b$$

LookupOp

mengkonversi nilai *pixel* demi *pixel* berdasarkan lookup table. Jika ditulis dalam bentuk persamaan matematik:

$$g(x, y) = T(f(x, y))$$

ConvolveOp

mendefinisikan operator konvolusi. Proses konvolusi adalah proses transformasi linear. Jika sebuah gambar direpresentasikan sebagai fungsi f(x, y), proses konvolusi ditulis sebagai:

$$g(x, y) = \iint K(x - u, y - v) f(u, v)\, du\, dv$$

di mana K adalah sebuah fungsi yang tetap, dikenal dengan istilah kernel. Properti dari operasi konvolusi ini ditentukan oleh fungsi kernel. Dengan memilih fungsi kernel yang tepat, Anda dapat memperoleh berbagai efek pada gambar, seperti smoothing, sharpening, dan edge detection.

Untuk gambar digital, interal berarti penjumlahan. Sehingga persamaan di atas dapat ditulis sebagai:

$$g(x, y) = \sum_i \sum_j K(x - i, y - j) f(i, j)$$

Nilai i dan j men-iterasikan seluruh *pixel* pada gambar. Untuk efisiensi, nilai K di-set 0 kecuali untuk tetangga titik pusat koordinat. Sebagai contoh, K(i,j) hanya memiliki 9 nilai bukan 0 ketika nilai -1<=i,j<=1. Pada kasus ini, persamaan di atas menjadi:

$$g(x, y) = \sum_{i=x-1}^{x+1} \sum_{j=y-1}^{y+1} K(x - i, y - j) f(i, j)$$

Pada setiap titik, hanya sembilan *pixel* di sekeliling titik yang perlu diperhitungkan dalam kalkulasi.

`ColorConvertOp`

mengkonversi warna *pixel* demi *pixel*.Operasi ini dilakukan dalam ruang warna.

`AffineTransformOp`

melakukan proses transformasi sebuah gambar. Operatornya tidak mengubah nilai *pixel*, tetapi memindahkan *pixel* tersebut ke lokasi yang berbeda. Object AffineTransform digunakan untuk men-set transformasinya. Dalam bentuk persamaan, operasi transformasi ini ditulis sebagai:

$$g(x, y) = f(A(x, y))$$

Pada Java 2D, untuk mengaplikasikan operator pada sebuah `BufferedImage`, lakukan pemanggilan *method* `filter` sebagai operator *object*.

```
dst = op.filter(src, null);
```

Sebuah `BufferedImage` dapat ditampilkan dengan *method* `drawImage` *object* `Graphics`. Sebagai contoh,

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawImage(bi, 0, 0, this);
}
```

Potongan program ini dapat dideklarasikan dalam komponen seperti `JPanel`. Pemanggilan *method* `drawImage` menggambar `BufferedImage` dimulai pada posisi (0,0).

3.15. Struktur Program Java2D

Berikut adalah struktur program yang dibangun dengan Java2D:

```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BasicSample2D extends JPanel{
5.    /* ATTRIBUTES*/
6.
7.    /* METHODS */
8.
9.    public BasicSample2D(){  //Constructor
10.
11.   }
12.
13.   public void clear(Graphics g){
14.       super.paintComponent(g);
15.   }
16.
17.   @Override
18.   public void paintComponent(Graphics g){
19.       clear(g);
20.
21.       // Cast Graphics to Graphics2D
22.      Graphics2D g2d = (Graphics2D)g;
23.
24.      // Set pen parameters
25.      g2d.setPaint(fillColorOrPattern);
26.      g2d setStroke(penThicknessOrPattern);
27.      g2d.setComposite(someAlphaComposite);
28.      g2d.setFont(anyFont);
29.      g2d.translate(...);
30.      g2d.rotate(...);
31.      g2d.scale(...);
32.      g2d.shear(...);
33.      g2d.setTransform(someAffineTransform);
34.
35.      // Create a Shape object
36.      SomeShape s = new SomeShape(...);
37.
38.      // Draw shape
39.      g2d.draw(s); // outline
40.      g2d.fill(s); // solid
41.   }
42.
43.   public static void main(String[] args){
44.       JFrame f = new JFrame("Basic Sample 2D");
45.       BasicSample2D bs = new BasicSample2D();
46.       f.getContentPane().add("Center",bs);
47.       f.pack();
48.       f.setSize(new Dimension(100,100));
49.       f.setVisible(true);
50.   }
51.}
```

Seperti dapat dilihat pada program, terdapat beberapa tahap penggambaran di Java2D:

1. Tahap 1: TYPECAST OBJECT GRAPHICS MENJADI GRAPHICS2D

   Class yang dibuat (BasicSample2D) meng-*extend class* JPanel (baris 4). Program grafik Java2D menggunakan class JPanel sebagai kanvas sebagai area untuk menggambar. Penggambaran dilakukan dengan meng-override *method* paintComponent dari kelas induknya dengan parameter masukan Graphics g (baris 18). Di dalam *body method*, parameter g ini kemudian di-*typecast* menjadi Graphics2D (baris 22). Di dalam *method* inilah semua fungsi grafik dan penggambaran dilakukan.

```
public void paintComponent(Graphics g) {
     super.paintComponent(g);
     Graphics2D g2d = (Graphics2D)g;
     g2d.doSomeStuff(...);
     ...
}
```

2. Tahap 2: SET PEN PARAMETER

   Proses yang dilakukan berikutnya adalah melakukan pengaturan parameter *pen*, tipe transformasi, dan *setting* awal lainnya (baris 24-33).

```
g2d.setPaint(WarnaAtauPola);
g2d.setStroke(KetebalanAtauPola);
g2d.setComposite(NilaiAlpha);
g2d.setFont(Font);
g2d.translate(...);
g2d.rotate(...);
g2d.scale(...);
g2d.shear(...);
g2d.setTransform(JenisTransformasi);
```

3. Tahap 3: CREATE SHAPE OBJECT

   Setelah setting dilakukan, *pen* siap digunakan untuk menggambar berbagai bentuk objek (baris 35-36).

```
Rectangle2D.Double rect = ...;
Ellipse2D.Double ellipse = ...;
Polygon poly = ...;
GeneralPath path = ...;
ShapeBentukan shape = ...;
```

4. Tahap 4: MENGGAMBAR SHAPE

   Gambar versi kerangka (*outlined*) atau bentuk penuh (*filled*) dari bentuk objek (baris 38-40).

```
g2d.draw(someShape);
g2d.fill(someShape);
```

5. Tahap 5: MAIN PROGRAM

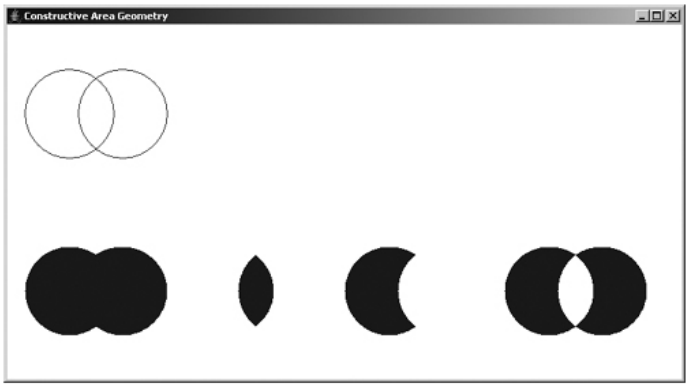
   Pada program utama, buat deklarasi dan *setting* awal untuk sebuah variabel bertipe JFrame sebagai arena untuk menggambar (baris 44-49).

BAB IV
IMPLEMENTASI DAN EKSPERIMEN

4.1. Constructive Area Geometry

Eksperimen mengenai teknik *constructive area geometry* dilakukan dengan menggabungkan dua buah lingkaran. Baris pertama pada gambar 4.1 menggambarkan dua bentuk pertama. Baris di bawahnya menggambarkan hasil kombinasi menggunakan keempat operasi (*add, intersect, subtract, exclusiveOr*) menghasilkan empat bentuk baru.

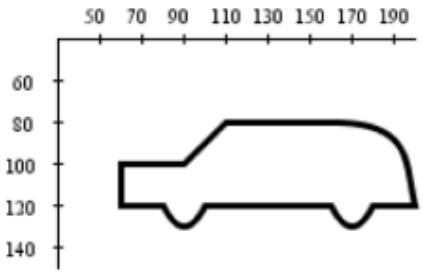
Potongan program dari eksperimen ini dapat dilihat di lampiran A.1.



Gambar 4.1. Hasil gambar dengan teknik *constructive area geometry*

4.2. General Path

Program pada lampiran A.2 menggambar bentuk mobil dengan menggunakan objek `GeneralPath`.



Gambar 4.2. Objek mobil dengan menggunakan GeneralPath

4.3. Bentuk Kurva Lain

Persamaan matematika berperan penting dalam memodelkan objek grafik. Sebaliknya, grafik dapat dijadikan sarana untuk mempelajari berbagai fungsi dan persamaan matematik.

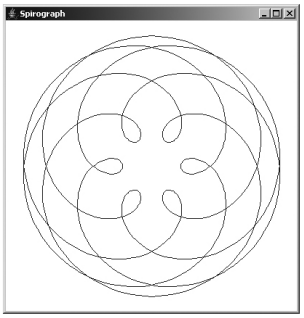
Cara yang paling sederhana untuk membentuk grafik dari sebuah persamaan adalah dengan menggambarkan sejumlah titik pada sistem koordinat yang memenuhi persamaan tersebut. Jadi, jika diberikan sebuah fungsi dengan bentuk y = f(x), maka pilih beberapa nilai koordinat x, kemudian hitung nilai koordinat y yang bersesuaian dengan masing-masing nilai x tersebut.

Program pada lampiran A.3 merupakan contoh dari pembentukan kurva berdasarkan persamaan:

$$x = (r_1 + r_2)\cos t - p.\cos\left(\frac{(r_1 + r_2)t}{r_2}\right)$$

$$y = (r_1 + r_2)\sin t - p.sin\left(\frac{(r_1 + r_2)t}{r_2}\right)$$

Nilai r_1 dan r_2 pada persamaan tersebut menyatakan jari-jari lingkaran pembentuk pola, sedangkan p menyatakan posisi relatif "pena" terhadap titik pusat kurva. Pola seperti ini dikenal dengan nama *epicycloid* atau *spirograph*. Pola yang terbentuk dapat dilihat pada gambar 4.3.



Gambar 4.3 Sebuah *spirograph*

Dari persamaan parametrik yang digunakan untuk membuat lingkaran dan elips, dapat dibuat bentuk-bentuk kurva lain yang menarik seperti pada contoh berikut :

1. Spiral

   Spiral dapat digambar menggunakan persamaan parametrik lingkaran dengan nilai r (radius) yang membesar sebanding dengan perbesaran sudut θ.

   Misalnya bentuk spiral di sebelah kiri dapat dibuat dengan persamaan parametrik :

$$r = 5 \,.\, \theta$$
$$x = x_c + r \,.\, cos(\theta)$$
$$y = y_c + r \,.\, sin(\theta)$$
$$0 <= \theta <= 6 \,.\, PI$$

Gambar 4.4 Kurva berbentuk spiral

2. Bentuk seperti daun

Bentuk seperti daun dapat dihasilkan dengan sebuah alat gambar (atau mungkin lebih cocok disebut mainan) berbentuk penggaris besar yang tengahnya memiliki lubang-lubang bergigi, dilengkapi bundaran-bundaran plastik yang juga bergigi dan memiliki lubang kecil yang pas untuk ujung pensil. Jika sebuah pensil di tempatkan di lubang kecil tadi, kemudian dilakukan gerakan memutar secara berulang-ulang, di atas kertas akan muncul pola-pola simetris yang menarik berbentuk seperti bunga atau daun-daun.

Dengan komputer, dapat dibentuk pola-pola seperti ini dengan lebih cepat. Caranya, adalah dengan menggunakan persamaan parametrik seperti yang digunakan untuk membuat lingkaran dengan sedikit modifikasi:

$x = x_c + r \,.\, cos(n \,.\, \theta) \,.\, cos(\theta)$

$y = y_c + r \,.\, cos(n \,.\, \theta) \,.\, sin(\theta); \quad 0 <= \theta <= 2 \,.\, PI$

Nilai n adalah suatu nilai konstanta tertentu di mana setiap nilai n yang berbeda akan membentuk pola yang berbeda pula:

Untuk n = 3 akan terbentuk pola kurva 3 helai daun seperti gambar pertama.

Untuk n = 2 pola berubah menjadi 4 helai daun.

Untuk n = 5 terbentuk kurva 5 helai daun seperti gambar ketiga.

Untuk n = 8 bentuknya menjadi 16 helai daun.

Gambar 4.5 Contoh berbagai pola daun yang dihasilkan

3. Kurva berbentuk seperti hati (*Cardiod*)

   Pola ini dibuat dengan memodulasikan jari-jari lingkaran dengan fungsi sinus.

$$x = x_c + (r + r \,.\, sin(\theta)) \,.\, cos(\theta)$$

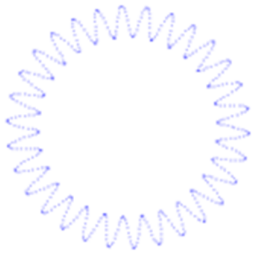
$$y = y_c + (r + r \,.\, sin(\theta)) \,.\, sin(\theta)$$

$$0 <= \theta <= 2 \,.\, PI$$

Gambar 4.6 Kurva *Cardiod*

4. Variasi bentuk lain

   Dengan sedikit eksperimen, bisa didapatkan bentuk kurva-kurva lain yang aneh dan menarik dari fungsi parametrik. Bentuk spiral melingkar seperti gambar 4.7 dapat dibuat dengan memodulasikan (menambah-kan) fungsi sinus (dengan periode berbeda) ke dalam fungsi parametrik lingkaran.



$$x = x_c + (r + 10 \,.\, sin(\Phi)) \,.\, cos(\theta)$$

$$y = y_c + (r + 10 \,.\, sin(\Phi)) \,.\, sin(\theta)$$

$$0 <= \theta <= 2 \,.\, PI$$

θ bertambah dengan pertambahan sebesar 1/r

Φ bertambah dengan pertambahan sebesar 30/r

Gambar 4.7 Kurva variasi bentuk

Bentuk helai daun yang lebih mirip daun sungguhan seperti gambar 4.8 dapat dibuat dengan memodulasikan fungsi sinus (dengan periode berbeda) ke dalam persamaan parametrik untuk membuat daun.

$$x = x_c + (r \,.\, cos(2 \,.\, \theta) + 10 \,.\, sin(\Phi)) \,.\, cos(\theta)$$

$$y = y_c + (r \,.\, cos(2 \,.\, \theta) + 10 \,.\, sin(\Phi)) \,.\, sin(\theta)$$

$$0 <= \theta <= 2 \,.\, PI$$

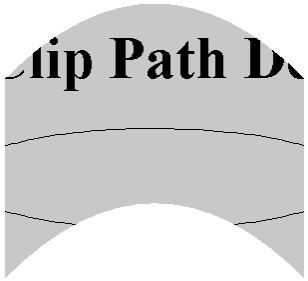
θ bertambah dengan pertambahan sebesar 1/r

Φ bertambah dengan pertambahan sebesar 30/r

Gambar 4.8 Kurva berbentuk daun

4.4. Clipping

Program untuk mendemokan penggunaan *clipping path* dapat dilihat di lampiran A.4. Program ini menggambar String “Clip Path Demo” dan sebuah oval ke layar, yang kemudian dipotong dengan suatu pola sehingga hanya bagian tertentu dari String dan oval tersebut yang terlihat di layar. Gambar berikut memperlihatkan hasil dari eksperimen *clipping*.



Gambar 4.9 Hasil *clipping* sebuah gambar

4.5. Teks & Font

Program untuk mendemokan fitur manipulasi font dan teks dari Java2D dapat dilihat di lampiran A.5. Program ini melakukan manipulasi teks untuk mendapatkan efek visual tertentu. Hal ini dapat dicapai melalui beberapa tahapan (lihat method `paintComponent`):

1. Objek `FontRenderContext` menspesifikasikan nilai-nilai yang dibutuhkan ketika menggambar font. Objek ini diakses melalui pemanggilan method `getFontRenderContext`. Pada contoh program di atas, digunakan objek font dengan spesifikasi: nama=”Serif”, style=”BOLD”, ukuran=144.
2. Seperti telah dijelaskan di atas, untuk memanipulasi karakter pada komputer grafik, kita membutuhkan objek glyph yang merupakan bentuk geometri dari karakter tertentu. Untuk mendapatkan objek `GlyphVector` dari suatu string, lakukan pemanggilan method `createGlyphVector` dari objek font. Method ini menerima dua parameter: Parameter pertama adalah objek `FontRenderContext`, parameter kedua adalah String yang akan ditampilkan. Program di atas menciptakan sebuah glyph dari String “Java”. Glyph ini disimpan dalam variabel bertipe `GlyphVector`.

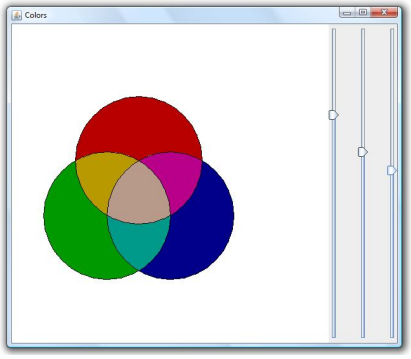
3. *Glyph* dari suatu karakter dengan jenis *font* tertentu, dapat dimanipulasi dengan mengubahnya menjadi objek `Shape`. Manipulasi ini dapat menghasilkan berbagai macam efek visual pada teks. Variabel `GlyphVector` yang telah diciptakan pada langkah sebelumnya di-*convert* menjadi objek Shape melalui pemanggilan *method* `getOutline`. Bentuk ini kemudian dijadikan sebagai *clipping path* pada proses penggambaran, melalui pemanggilan *method* `setClip` dari objek Graphics2D.
4. Dua ribu elips digambar dengan lokasi acak pada panel. Hanya elips-elips yang berada di dalam glyph saja yang terlihat pada layar. String teks "Java" tidak secara eksplisit ditulis pada layar, tetapi hanya dijadikan sebagai *clipping path* dari gambar. Hal ini yang membuat tulisan "Java" menjadi menonjol.

Gambar 4.10 memperlihatkan hasil *run* program di lampiran A.5.

Gambar 4.10 Contoh manipulasi teks dalam Java2D

4.6. Warna

Eksperimen berikut mengilustrasikan kombinasi ketiga warna primer (RGB). Program pada lampiran A.6 menggambar 3 lingkaran yang saling beririsan. Tujuh daerah berbeda merepresentasikan kombinasi dari ketiga warna. Tiga buah slider di sebelah kanan mengatur nilai komponen *red*, *green*, dan *blue*.



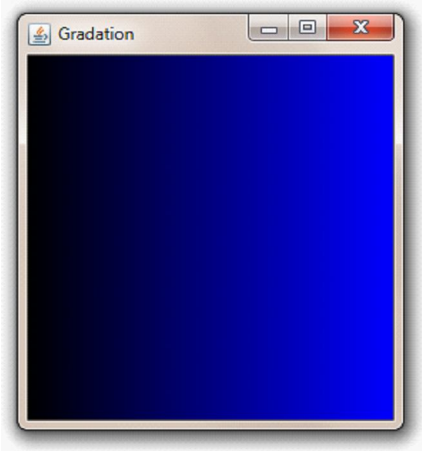
Gambar 4.11 Program untuk mendemokan warna dan kombinasinya

4.7. Paint

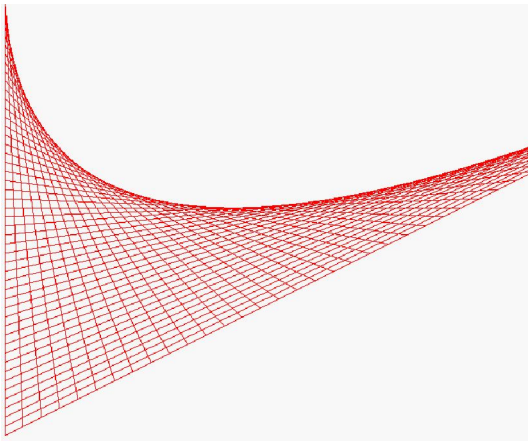
Selain dengan komposisi warna merah, hijau, dan biru yang homogen, sebuah bentuk geometri juga dapat diwarnai dengan pola tertentu atau diwarnai dengan gambar. Eksperimen pada lampiran A.7, dibentuk tiga buah bentuk geometri (segiempat, oval, dan teks “Java”) yang diwarnai dengan gambar atau warna yang tidak homogen, tetapi memiliki pola tertentu.

Gambar 4.12 Eksperimen mewarnai bentuk geometri dengan gambar & warna berpola

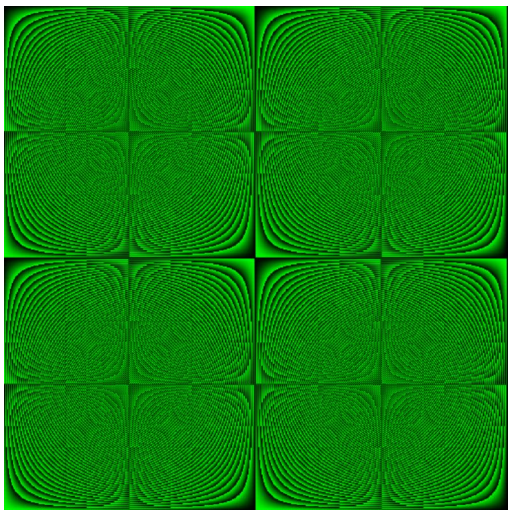
Hanya dengan menggunakan titik dan warna dapat dibuat berbagai pola-pola menarik. Pada eksperimen titik dan warna berikut, dapat dibentuk gradasi warna (lampiran A.8), pola jaring, pola unik yang dikenal dengan “Moire Pattern” (lampiran A.9) seperti dapat dilihat pada gambar 4.13 sampai dengan gambar 4.15.



Gambar 4.13 Permainan titik dan warna untuk menghasilkan gradasi warna



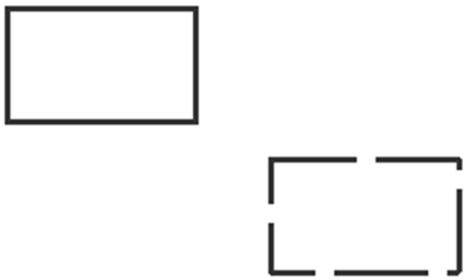
Gambar 4.14. Permainan titik dan warna untuk menghasilkan pola jaring



Gambar 4.15 Moire Pattern

4.8. Transformasi: Translasi

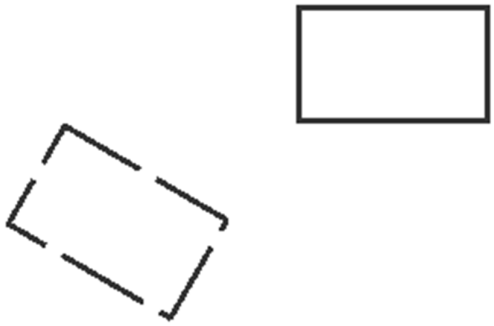
Program pada lampiran A.10 menggeser sebuah segiempat sebanyak 140 dalam arah x, 80 dalam arah y. Segiempat dengan garis putus-putus merupakan hasil translasi segiempat asal pada gambar 4.16.



Gambar 4.16 Translasi segiempat 140 dalam arah x, 80 dalam arah y

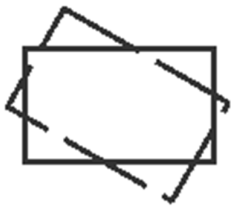
4.9. Transformasi: Rotasi

Program pada lampiran A.11 merotasikan sebuah segiempat sebesar 30^0. Gambar 4.17 merupakan hasil yang ditampilkan jika program tersebut dijalankan. Segiempat dengan garis putus-putus merupakan hasil rotasi segiempat asal.



Gambar 4.17 Hasil rotasi mengubah posisi objek

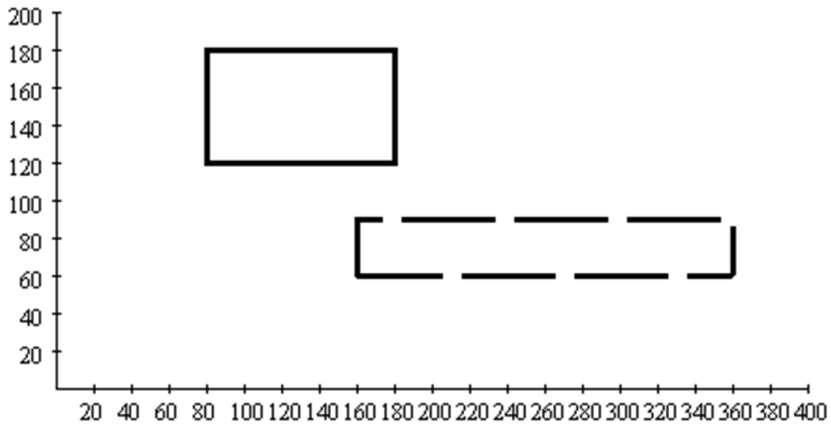
Seperti terlihat pada gambar, proses rotasi mengubah posisi objek. Agar hasil rotasi tidak mengubah posisi objek, perlu dilakukan pendefinisian titik pusat objek sebagai titik pusat rotasi. Gambar 4.18 merupakan contoh hasil rotasi dengan titik pusat objek sebagai titik pusat rotasi.



Gambar 4.18 Hasil rotasi dengan titik pusat objek sebagai pusat rotasi

4.10. Transformasi: Penskalaan

Potongan program pada lampiran A.12 menskalakan sebuah segiempat sebanyak 2 kali dalam arah x dan 0.5 dalam arah y (*differential scaling*). Gambar 4.19 adalah hasil *run* program ini. Segiempat dengan garis putus-putus merupakan hasil penskalaan segiempat awal.



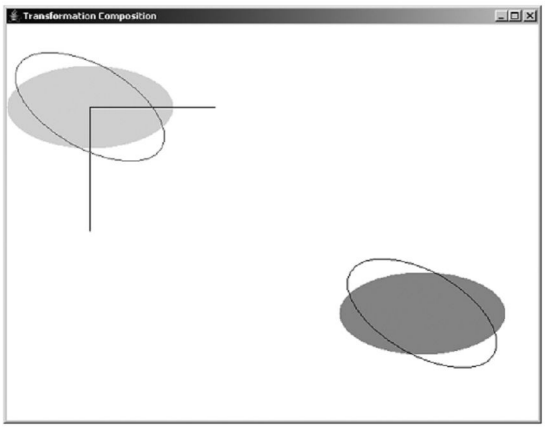
Gambar 4.19 Contoh *differential scaling*

4.11. Komposisi Transformasi

Program pada lampiran A.13 merotasikan sebuah elips terhadap titik pusat koordinat dengan komposisi transformasi translasi dan rotasi. Komposisi transformasi ini dilakukan agar hasil rotasi tidak mengubah kedudukan awal objek. Objek hasil rotasi akan tetap berada pada posisi awalnya. Kedudukan objek dapat berubah dari posisi awalnya jika hanya dilakukan rotasi tanpa translasi.

Program ini menggambar bentuk yang berbeda pada setiap tahap komposisi transformasi. Pertama, digambar sebuah elips dengan *bounding rectangle* (300, 200, 200, 100), sehingga titik pusatnya berada di (400, 250). Tujuan akhir program ini adalah merotasikan elips tersebut sebesar 30^0 terhadap titik pusatnya (400,250). Untuk mencapai hal ini, digunakan komposisi yang terdiri dari tiga buah transformasi. Transformasi pertama adalah translasi sebesar (-400, -250) untuk memindahkan elips sehingga titik pusatnya berada di (0,0). Transformasi kedua adalah rotasi sebesar 30^0 terhadap titik pusat (0,0). Terakhir, elips ini dipindahkan kembali ke posisi awalnya dengan transformasi translasi sehingga titik pusatnya kembali berada di posisi (400, 250). Dengan komposisi ketiga transformasi ini, tercapailah rotasi seperti yang diharapkan, yaitu hasil rotasi tetap berada pada posisi awalnya. Elips yang terbentuk pada setiap tahap transformasi ditampilkan dengan tingkat warna keabuan yang berbeda. Elips hasil rotasi ditampilkan dalam bentuk kerangka (tanpa warna bidang dalam).

Gambar mengilustrasikan proses komposisi transformasi yang dilakukan.



Gambar 4.20 Proses rotasi elips terhadap titik pusat objek

4.12. Animasi Sederhana

Program pada lampiran A.14 menganimasikan gerakan sebuah bola yang seolah-olah jatuh dari ketinggian tertentu kemudian terpental lagi ke atas dan seterusnya sampai berhenti. Ada 2 class untuk program ini: 1. Class Drop2DApplet dan 2. Class Drop2Dpanel.java.

4.13. Image Processing

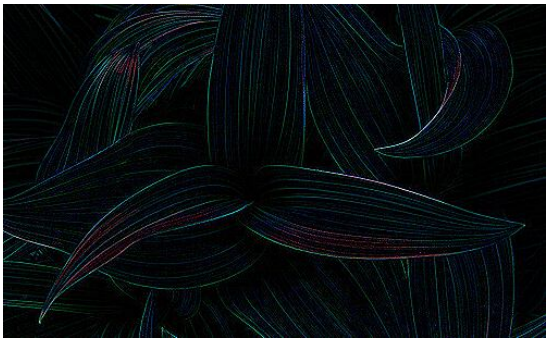
Pada eksperimen pengolahan citra ini (lampiran A.15), *user* dapat me-*load* gambar dari disk file, melakukan operasi pengolahan citra, kemudian menyimpan hasilnya pada disk. Gambar berikut merupakan contoh gambar asal yang akan diproses:

Gambar 4.21 Gambar asal yang akan diproses

Hasil operasi *smooth*, *sharpen*, *edge*, dan *grayscale* dari gambar 4.21 dapat dilihat pada gambar 4.22-4.25.

Gambar 4.22 Contoh hasil operasi *smooth*

Gambar 4.23 Contoh hasil operasi *sharpen*



Gambar 4.24 Contoh hasil operasi *edge*

Gambar 4.25 Contoh hasil operasi *grayscale*

Operasi-operasi pemrosesan gambar yang dibuat pada program di atas di antaranya: *smooth, sharpen, edge detection, rescale, rotation,* dan *grayscale*.

Operasi *smooth* pada program di atas adalah proses konvolusi oleh kernel 3x3:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Jumlah seluruh elemen matriks adalah 1 (9 x 1/9). Pada Java2D, sebuah kernel adalah array bertipe float berukuran 2D. Pada program memberi tahu *instance class* `Kernel` untuk "memperlakukan" array `data` sebagai matriks berukuran 3x3.

Operator *sharpen* didefinisikan dengan oleh kernel 3x3:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Operator *edge-detection* didefinisikan oleh kernel 3x3:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Operasi rescale menggunakan class `RescaleOp.` Operasi ini digunakan untuk menambah pencahayaan gambar.

Operasi rotation menggunakan class `AffineTransformOp`. Rotasi sebesar p/6 didefinisikan dengan objek `AffineTransform` kemudian diaplikasikan pada objek `AffineTransformOp`.

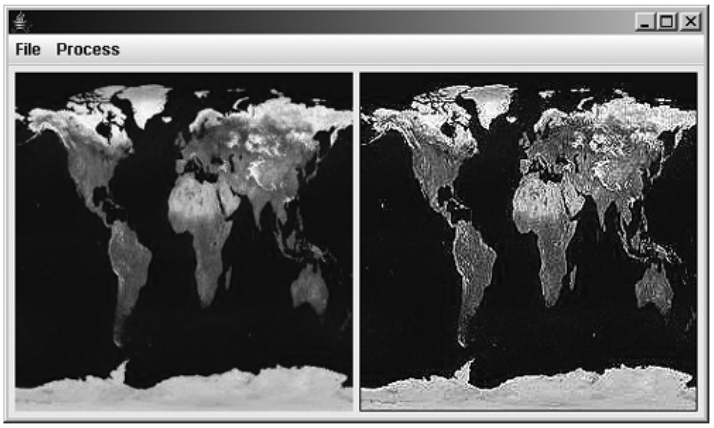
Operasi grayscale menggunakan class `ColorConvertOp` untuk mengubah gambar asal ke gambar *grayscale*.

Operasi *copy* menyalin gambar hasil pemrosesan (sebelah kanan) menjadi gambar asal (sebelah kiri), sehingga gambar ini dapat diproses lagi dengan operator pengolahan citra yang lain.

Gambar yang telah diproses dapat disimpan pada *disk file* dengan format PNG. `JFileChooser` digunakan untuk memilih *file* yang akan ditulis. *Static method* `save` dari *class* `ImageIO` digunakan untuk menyimpan gambar.

Program ini menyediakan dua menu: *File* dan *Process.* Pada menu *File,* terdiri dari beberapa sub-menu: membuka dan menyimpan *file* gambar, keluar dari program. Menu *Process* terdiri dari berbagai pilihan operasi pengolahan citra. Di bagian bawah menu,

terdapat dua buah objek gambar. Gambar sebelah kiri adalah gambar asal. Gambar sebelah kanan adalah gambar hasil pengolahan citra *sharpening*.



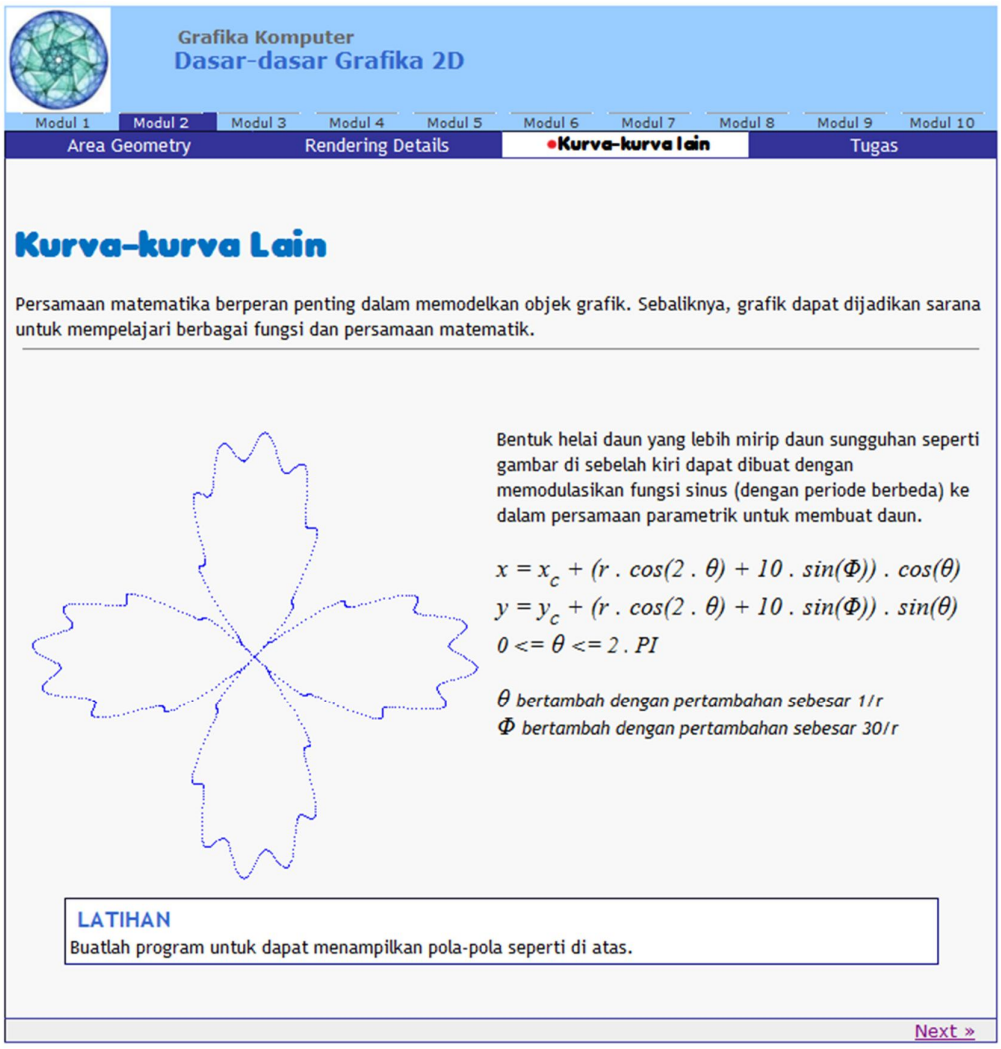
Gambar 4.26 Hasil *run* program *image processing*

Class `ImagePanel` meng-extend *class* `JPanel` digunakan untuk menampilkan `BufferedImage`. Sebuah gambar dapat dikirimkan pada `ImagePanel` melalui pemanggilan *constructor* atau *method* `setImage`. Program mengimplementasikan dua cara proses *loading file* gambar. Metode pertama menggunakan *static method* `read` dari class `ImageIOPanel` untuk me-load file gambar secara langsung ke objek `BufferedImage`. Metode kedua menggunakan AWT *image loading*, `BufferedImage` diperoleh dengan menggambar citra. Objek `JFileChooser` memungkinkan *user* memilih *file* gambar untuk dibuka.

4.14. Pembangunan Modul

Seluruh materi yang disampaikan di atas mempunyai bentuk modul berbasis *web* sehingga memudahkan penelusuran. Selain berisi penjelasan materi, modul ini juga memberikan contoh-contoh dan latihan-latihan kecil. Materi-materi di atas dikelompokkan dan dibagi ke dalam beberapa modul untuk memudahkan pemahaman. Salah satu contoh antarmuka modul dapat dilihat pada gambar 4.27. Beberapa bagian dari halaman modul tersebut dihilangkan untuk kepentingan memperkecil gambar. Bagian yang dihilangkan tersebut di antaranya bagian penjelasan materi dan listing program. Penjelasan materi disampaikan secara bertahap seperti yang disampaikan di bab 3 buku ini. Listing program cukup panjang, potongan kode-kode program penting disampaikan pada lampiran A.

Seperti terlihat pada gambar 4.27, ketika *user* berada pada suatu modul (dalam contoh ini modul 2), *user* dapat melihat modul-modul lain dengan mengklik *link* ke modul-modul lain yang terdapat pada bagian *header* dari web. *User* dapat membaca penjelasan materi, mencoba contoh program-program kecil, bereksperimen melalui pertanyaan-pertanyaan dan latihan-latihan kecil yang terdapat dalam modul.



Gambar 4.27. Contoh antarmuka modul eksplorasi Java2D

BAB V
KESIMPULAN

Dari hasil eksplorasi Java2D, dapat disimpulkan bahwa:

1. Sistem grafik 2D memodelkan *virtual world* dalam dunia 2 dimensi. Java2D menyediakan fungsi-fungsi yang lengkap untuk penggambaran grafik 2 dimensi, mulai dari penggambaran bentuk geometri dasar, teks, fungsi manipulasi grafik, fungsi *rendering* 2D, sampai dengan fungsi-fungsi yang dapat digunakan untuk penggambaran yang lebih kompleks seperti pemrosesan gambar.
2. Java2D merupakan pilihan yang tepat untuk mempelajari pemrograman grafik dengan bahasa Java.

LAMPIRAN A

KODE SUMBER

A.1 Constructive Area Geometri

```
public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    Shape s1 = new Ellipse2D.Double(0, 0, 100, 100);
    Shape s2 = new Ellipse2D.Double(60, 0, 100, 100);
    g2.translate(20, 50);
    g2.draw(s1);
    g2.draw(s2);
    g2.translate(0,200);

    Area a1 = new Area(s1);
    Area a2 = new Area(s2);
    a1.add(a2);
    g2.fill(a1);

    g2.translate(180,0);
    a1 = new Area(s1);
    a1.intersect(a2);
    g2.fill(a1);

    g2.translate(180,0);
    a1 = new Area(s1);
    a1.subtract(a2);
    g2.fill(a1);

    g2.translate(180,0);
    a1 = new Area(s1);
    a1.exclusiveOr(a2);
    g2.fill(a1);
}
```

A.2 General Path

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.setColor(Color.blue);
    GeneralPath gp = new GeneralPath();
    gp.moveTo(60,120);
    gp.lineTo(80,120);
    gp.quadTo(90,140,100,120);
    gp.lineTo(160,120);
    gp.quadTo(170,140,180,120);
    gp.lineTo(200,120);
    gp.curveTo(195,100,200,80,160,80);
    gp.lineTo(110,80);
    gp.lineTo(90,100);
    gp.lineTo(60,100);
    gp.lineTo(60,120);
    g2.draw(gp);
}
```

A.3 Bentuk Kurva Lain

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.translate(200,200);
    int x1 = (int)(r1 + r2 - p);
    int y1 = 0;
    int x2;
    int y2;
    for (int i = 0; i < nPoints; i++) {
        double t = i * Math.PI / 90;
        x2 = (int)((r1+r2)*Math.cos(t)-p*Math.cos((r1+r2)*t/r2));
        y2 = (int)((r1+r2)*Math.sin(t)-p*Math.sin((r1+r2)*t/r2));
        g2.drawLine(x1, y1, x2, y2);
        x1 = x2;
        y1 = y2;
    }
}
```

A.4 Clipping

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    GeneralPath path = new GeneralPath(GeneralPath.WIND_EVEN_ODD);
    path.moveTo(100,200);
    path.quadTo(250, 50, 400, 200);
    path.lineTo(400,400);
    path.quadTo(250,250,100,400);
    path.closePath();
    g2.clip(path);
    g2.setColor(new Color(200,200,200));
    g2.fill(path);
    g2.setColor(Color.black);
    g2.setFont(new Font("Serif", Font.BOLD, 60));
    g2.drawString("Clip Path Demo",80,200);
    g2.drawOval(50, 250, 400, 100);
}
```

A.5 Teks & Font

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    Font font = new Font("Serif", Font.BOLD, 144);
    FontRenderContext frc = g2.getFontRenderContext();
    GlyphVector gv = font.createGlyphVector(frc, "Java");
    Shape glyph = gv.getOutline(100,200);
    g2.setClip(glyph);
    g2.setColor(Color.red);
    for (int i = 0; i < 2000; i++) {
        Shape shape = new Ellipse2D.Double(Math.random()*500,
        Math.random()*400, 30, 20);
        g2.draw(shape);
    }
}
```

A.6 Warna

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    Shape rc = new Ellipse2D.Double(100, 113, 200, 200);
    Shape gc = new Ellipse2D.Double(50, 200, 200, 200);
    Shape bc = new Ellipse2D.Double(150, 200, 200, 200);
    Area ra = new Area(rc);
    Area ga = new Area(gc);
    Area ba = new Area(bc);
    Area rga = new Area(rc);        rga.intersect(ga);
    Area gba = new Area(gc);        gba.intersect(ba);
    Area bra = new Area(bc);        bra.intersect(ra);
    Area rgba = new Area(rga);      rgba.intersect(ba);
    ra.subtract(rga);       ra.subtract(bra);
    ga.subtract(rga);       ga.subtract(gba);
    ba.subtract(bra);       ba.subtract(gba);
    g2.setColor(new Color(red,0,0));        g2.fill(ra);
    g2.setColor(new Color(0,green,0));      g2.fill(ga);
    g2.setColor(new Color(0,0,blue));       g2.fill(ba);
    g2.setColor(new Color(red,green,0));    g2.fill(rga);
    g2.setColor(new Color(0,green,blue));   g2.fill(gba);
    g2.setColor(new Color(red,0,blue));     g2.fill(bra);
    g2.setColor(new Color(red,green,blue)); g2.fill(rgba);

    g2.setColor(Color.black);
    g2.draw(rc);        g2.draw(gc);        g2.draw(bc);
}
```

A.7 Paint

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    GradientPaint gp = new GradientPaint(100,50,Color.white, 150, 50, Color.gray, true);
    g2.setPaint(gp);
    g2.fillRect(100, 40, 300, 20);
    TexturePaint tp = new TexturePaint(image,
            new Rectangle2D.Double(90, 90, image.getWidth(),image.getHeight()));
    g2.setPaint(tp);
    Shape ellipse = new Ellipse2D.Double(110, 110, 250, 200);
    g2.fill(ellipse);
    GradientPaint paint = new GradientPaint(100, 300, Color.white,400, 400, Color.black);
    g2.setPaint(paint);
    Font font = new Font("Serif", Font.BOLD, 144);
    g2.setFont(font);
    g2.drawString("Java", 100, 400);
}
```

A.8 Gradasi Warna

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;

    for(int x=0;x<256;x++) {
        for(int y=0;y<256;y++) {
            g2.setColor(new Color(0,0,x));
            g2.drawLine(x,y,x,y);
        }
    }
}
```

A.9 Moire Pattern

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;

    for(int x=0;x<768;x++) {
        for(int y=0;y<768;y++) {
            g2.setColor(new Color(0,(x*y)%256,0));
            g2.drawLine(x,y,x,y);
        }
    }
}
```

A.10 Transformasi: Translasi

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;

    //Ketebalan garis diset menjadi 3
    g2d.setStroke(new BasicStroke(3.0f));

    //Generate dan gambar segiempat yang akan ditransfomasi
    Rectangle2D.Double rect = new Rectangle2D.Double(20,20,100,60);
    g2d.draw(rect);

    //Definisi transformasi (translasi)
    AffineTransform translation = new AffineTransform();
    translation.setToTranslation(140,80);

    //Gambar segiempat hasil translasi dengan garis putus-putus
    g2d.setStroke(new BasicStroke(3.0f,BasicStroke.CAP_BUTT,
                                  BasicStroke.JOIN_BEVEL,8.0f,
                                  new float[] {50.0f, 10.0f},4.0f));
    g2d.draw(translation.createTransformedShape(rect));
}
```

A.11 Transformasi: Rotasi

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;

    //Ketebalan garis diset menjadi 3
    g2d.setStroke(new BasicStroke(3.0f));

    //Generate dan gambar segiempat yang akan ditransfomasi
    Rectangle2D.Double rect = new Rectangle2D.Double(180,200,100,60);
    g2d.draw(rect);

    //Definisi transformasi (rotasi)
    AffineTransform rotating= new AffineTransform();
    rotating.setToRotation(Math.PI/6);

    //Gambar segiempat hasil penskalaan dengan garis putus-putus
    g2d.setStroke(new BasicStroke(3.0f,BasicStroke.CAP_BUTT,
                                  BasicStroke.JOIN_BEVEL,8.0f,
                                  new float[] {50.0f, 10.0f},4.0f));
    g2d.draw(rotating.createTransformedShape(rect));
}
```

A.12 Transformasi: Penskalaan

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;

    //Ketebalan garis diset menjadi 3
    g2d.setStroke(new BasicStroke(3.0f));

    //Generate dan gambar segiempat yang akan ditransfomasi
    Rectangle2D.Double rect = new Rectangle2D.Double(80,120,100,60);
    g2d.draw(rect);

    //Definisi transformasi (penskalaan)
    AffineTransform scaling = new AffineTransform();
    scaling.setToScale(2,0.5);

    //Gambar segiempat hasil penskalaan dengan garis putus-putus
    g2d.setStroke(new BasicStroke(3.0f,BasicStroke.CAP_BUTT,
                                  BasicStroke.JOIN_BEVEL,8.0f,
                                  new float[] {50.0f, 10.0f},4.0f));
    g2d.draw(scaling.createTransformedShape(rect));
}
```

A.13 Komposisi Transformasi

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    g2.translate(100,100);
    Shape e = new Ellipse2D.Double(300, 200, 200, 100);
    g2.setColor(new Color(160,160,160));
    g2.fill(e);
    AffineTransform transform = new AffineTransform();
    transform.translate(-400,-250);
    e = transform.createTransformedShape(e);
    g2.setColor(new Color(220,220,220));
    g2.fill(e);
    g2.setColor(Color.black);
    g2.drawLine(0, 0, 150, 0);
    g2.drawLine(0, 0, 0, 150);
    transform.setToRotation(Math.PI / 6.0);
    e = transform.createTransformedShape(e);
    //g2.setColor(new Color(100,100,100));
    g2.setColor(Color.red);
    g2.draw(e);
    transform.setToTranslation(400, 250);
    e = transform.createTransformedShape(e);
    g2.setColor(new Color(0,0,0));
    g2.draw(e);
}
```

A.14 Animasi Sederhana

```
public void stop () {
    // Men-set thread ke null membuat loop pada
    // method run() berhenti & membuang thread.
    fThread = null;
}

/** Pengulangan thread untuk menggambar setiap frame animasi. **/
public void run () {
    // Disable button selama animasi drop
    fDropButton.setEnabled (false);

    // Inisialisasi keadaan bola sebelum animasi
    fDropPanel.reset ();

    // Loop animasi
    while ( fThread != null){
        // Sleep 25msecs antar frame
        try{ Thread.sleep (25);
        } catch  (InterruptedException e) { }
        // Repaint drop panel untuk setiap frame baru
        fDropPanel.repaint ();
        if (fDropPanel.isDone ()) fThread = null;
    }

    // Enable button untuk animasi yang baru
    fDropButton.setEnabled (true);
  }
```

A.14 Animasi Sederhana

```
/** Draw the ball at its current position. **/
public void paintComponent (Graphics g){
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;

    // Hitung posisi seiring waktu
    calcPosition ();
    // Pindahkan posisi bola.
    fBall.setFrame(fXPixel-fRadius,fYPixel-fRadius,fDiam,fDiam);
    g2.setColor (Color.RED);
    g2.fill(fBall);
    g2.draw (fBall);
}

/** Menghitung posisi bola pada frame berikut. **/
void calcPosition () {
    // Increment by 10 millseconds per frame
    double dt = 0.025;

    // Menghitung posisi dan kecepatan setiap saat
    fY = fY + fVy * dt - 490.* dt * dt;
    fVy = fVy  -  980.0 * dt;

    // Konversi ke satuan pixel
    fYPixel = fFrameHt - (int)(fY * fYConvert);

    // Membalik arah gerak bola ketika mencapai dasar
    if ((fYPixel + fRadius) >= (fFrameHt-1)) {
        fVy = Math.abs (fVy);
        // Kurangi sedikit kecepatannya
        fVy -= 0.1*fVy;
        // Hentikan bola ketika mencapai kecepatan tertentu
        if (fVy < 15.0) {
            fDropDone=true;
        }
    }
}
```

A.15 Image Processing

```
void process(String opName) {
    BufferedImageOp op = null;
    if (opName.equals("Smooth")) {
        float[] data = new float[9];
        for (int i = 0; i < 9; i++) data[i] = 1.0f/9.0f;
        Kernel ker = new Kernel(3,3,data);
        op = new ConvolveOp(ker);
    } else if (opName.equals("Sharpen")) {
    float[] data = {0f, -1f, 0f, -1f, 5f, -1f, 0f, -1f, 0f};
    Kernel ker = new Kernel(3,3,data);
    op = new ConvolveOp(ker);
    } else if (opName.equals("Edge")) {
        float[] data = {0f, -1f, 0f, -1f, 4f, -1f, 0f, -1f, 0f};
        Kernel ker = new Kernel(3,3,data);
        op = new ConvolveOp(ker);
    } else if (opName.equals("Rescale")) {
        op = new RescaleOp(1.5f, 0.0f, null);
    } else if (opName.equals("Gray scale")) {
        op = new ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
    } else if (opName.equals("Rotate")) {
        AffineTransform xform = new AffineTransform();
        xform.setToRotation(Math.PI/6);
        op = new AffineTransformOp(xform, AffineTransformOp.TYPE_BILINEAR);
    }
    BufferedImage bi = op.filter(imageSrc.getImage(), null);
    imageDst.setImage(bi);
    pack();
}
```

DAFTAR PUSTAKA

[1] Klawonn F. *Introduction to Computer Graphics using Java2D and 3D*. Springer. 2008.

[2] Knudsen J. *Java 2D Graphics*. O'Reilly. 1999.