Javaアプリケーションでのイメージ作成

低負荷で、簡単なグラフィック・イメージの描画とレンダリングを行う

Ivor Horton

2001年 2月 01日

この記事は、Beginning Java 2 -- JDK 1.3 Edition (Wrox Press、2000年3月) からの抜粋です。著者Ivor Horton氏が、独自のグラフィック・ファイルを作成する場合の基本事項を説明します。スプライトのレンダリング、アルファ・チャネルの操作、最高の品質を実現するヒントを紹介します。

イメージの合成

ファイルからすべてのイメージを読み込む必要はありません。独自のイメージを作成することができます。これを最も柔軟に行うためには、`BufferedImage` オブジェクトを使用する必要があります。このオブジェクトは、アクセス可能なバッファーにイメージ・データが格納されている`Image` クラスのサブクラスです。また、アルファ・チャネルの有無、異なるタイプのカラー・モデルの使用、さまざまな精度のカラー・コンポーネントの使用など、ピクセル・データを格納する各種方法もサポートしています。`ColorModel` クラスでは、`BufferedImage` オブジェクトで使用するさまざまな種類のカラー・モデルを柔軟に定義することができます。カラー・モデルの基本的な動作を理解するために、ここでは、カラー・コンポーネントがRGB値であるデフォルトの1つのカラー・モデルと、8ビットのRGBカラー値とアルファ・チャネルを格納する1つのバッファー・タイプだけを使用します。このバッファー・タイプは、`BufferedImage` クラスで、定数`TYPE_INT_ARGB` によって指定します。これにより、各ピクセルに`int` 値を使用することが暗黙指定されます。各ピクセルの値には、アルファ・コンポーネントと、8ビット・バイトとしてのRGBカラー・コンポーネントが格納されます。このタイプの`BufferedImage` オブジェクトは、次のような文を使って幅と高さを指定して作成することができます。

```
int width = 200;
int height = 300;
BufferedImage image = new BufferedImage(width,
 height,BufferedImage.TYPE_INT_ARGB);
```

このコードは、幅200ピクセル、高さ300ピクセルのイメージを表す`BufferedImage` オブジェクトを作成します。このイメージの上に描画するには、グラフィックス・コンテキストが必要なので、`BufferedImage`オブジェクトの`createGraphics()` メソッドが、イメージに関連する`Graphics2D` オブジェクトを戻します。

```
int width = 200;
Graphics2D g2D = image.createGraphics();
```

g2D オブジェクトを使った操作により、BufferedImage オブジェクト、すなわちイメージ内のピクセルを変更します。このオブジェクトを利用できれば、BufferedImage オブジェクトの上で描画を行う完全な機能が得られます。形状、イメージ、GeneralPath オブジェクトなど何でも描画することができ、グラフィックス・コンテキストのアルファ・コンポジット・オブジェクトを設定できます。また、Graphics2D オブジェクトに付属の完全なアフィン変換機能も得られます。

BufferedImage オブジェクトの個々のピクセルを取得するには、getRGB() メソッドを呼び出して、ピクセルのx,y座標を型int の引き数として指定します。ピクセルは、4つの8ビット値で表されたアルファと、32ビット・ワードにパックされたRGBカラー・コンポーネントからなるTYPE_INT_ARGB 形式で型int として戻されます。また、イメージ・データの一部からピクセルの配列を戻す多重定義バージョンのgetRGB() もあります。個々のピクセル値を設定するには、setRGB() メソッドを呼び出します。最初の2つの引き数は、ピクセルの座標で、3つ目の引き数は、型をint の設定する値です。また、このメソッドには、ピクセルの配列の値を設定するバージョンもあります。

これで、ピクセルの操作は終了です。次に、Wroxロゴの背景でBufferedImage オブジェクトのアニメーションを作成するアプレットを作成します。この例では、イメージの一部を透過にする方法も示します。アプレット・ファイルの基本的な内容は次のとおりです。

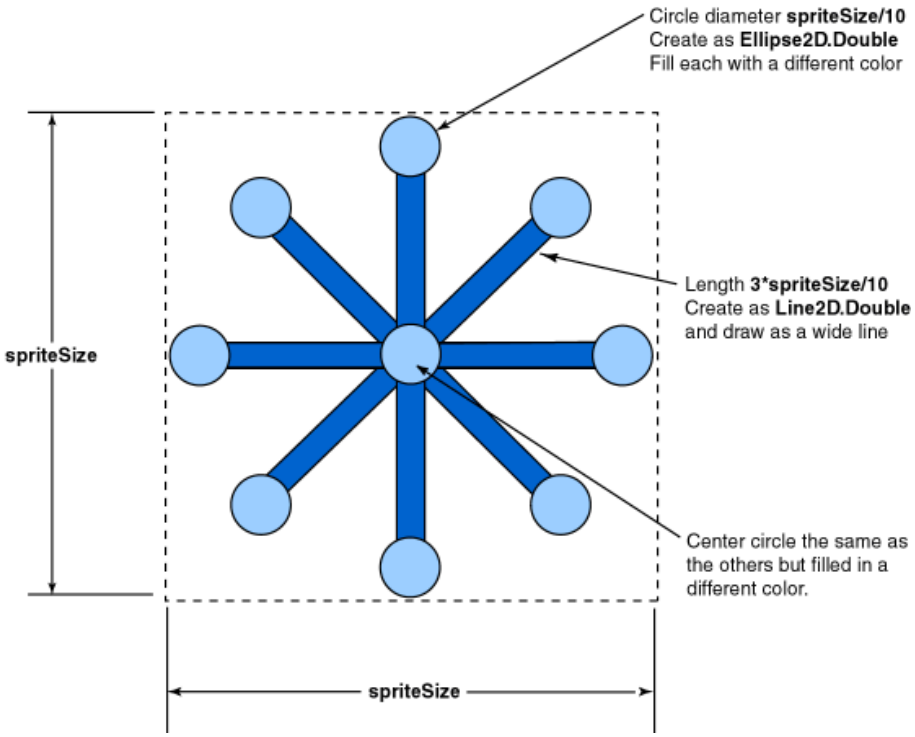
```
import java.awt.*;
import java.awt.image.*;
import java.awt.geom.*;
import javax.swing.*;
public class ImageDrawDemo extends JApplet
{
  // The init() method to initialize everything...
  // The start() method to start the animation...
  // The stop() method to stop the animation...
  // The ImagePanel class defining the panel displaying the animation...
  // Data members for the applet...
}
```

イメージの作成

スプライト は、アニメーションを作成する際に静的イメージの上に描画できる小さいグラフィカル・イメージです。アニメーション効果を出す場合は、スプライトを一定の時間、位置や方向を変えて描画します。もちろん、これを簡単にするには、座標系の変換が非常に役立ちます。スプライトはゲームでよく使用されます。スプライトを静的な背景に対して描画するだけなので、アニメーションにプロセッサー時間をあまりかけずに済ませることができます。BufferedImage オブジェクトの使用に関心があるのは、プロセッサー時間を最小限に抑える最善の方法にひかれているからではありません。ここでの目的は、プログラムの内部でイメージを作成し、使用できる方法を理解することです。

BufferedImage オブジェクトは、図1に示すイメージのようになります。

図1. **BufferedImage** のスプライト



このイメージは、辺の長さを`spriteSize`とする正方形です。イメージの他の部分の寸法は、この長さを基準としています。実際、ここには、ジオメトリック・エンティティーは線と円の2つしかありません。各ジオメトリック・エンティティーは、位置や方位を変えて繰り返されています。線に対して`Line2D.Double`オブジェクトを作成し、円に対して`Ellipse2D.Double`オブジェクトを作成した場合は、ユーザー座標系を移動して、これらの2つのオブジェクトのうちの一方を描画することで、全体を描画できなければなりません。

純粋なオブジェクト指向のアプローチでは、スプライトを表すクラスは、おそらく`BufferedImage`のサブクラスとして定義しますが、ここでは、`BufferedImage`オブジェクトの使用法を扱うので、目的に従って`BufferedImage`オブジェクト上にスプライトを描画するメソッド`createSprite()`を開発することにします。このメソッドは、アプレット・クラスの単なるメンバーなので、アプレットにデータ・メンバーを追加して、必要なデータを格納します。使用するデータ・メンバーをアプレット・クラスのアウトラインに接続することができます。

```
double totalAngle; // Current angular position of sprite
double spriteAngle; // Rotation angle of sprite about its center
ImagePanel imagePanel; // Panel to display animation
BufferedImage sprite; // Stores reference to the sprite
int spriteSize = 100; // Diameter of the sprite
Ellipse2D.Double circle; // A circle - part of the sprite
Line2D.Double line; // A line - part of the sprite
// Colors used in sprite
Color[] colors = {Color.red , Color.yellow, Color.green , Color.blue,
Color.cyan, Color.pink , Color.magenta, Color.orange};
java.util.Timer timer; // Timer for the animation
long interval = 50; // Time interval msec between repaints
```

これらのメンバーの一般的な使い方は、コメントと切り離して考えなければなりません。コードを作成しながら、その使い方を紹介しましょう。

createSprite() メソッドが最初に実行しなければならないことは、BufferedImage オブジェクトであるsprite の作成です。sprite イメージを作成するには、Graphics2D オブジェクトが必要です。この場合のコードは次のようになります。

```
BufferedImage createSprite(int spriteSize)
{
  // Create image with RGB and alpha channel
BufferedImage sprite = new BufferedImage(spriteSize, spriteSize, BufferedImage.TYPE_INT_ARGB);
Graphics2D g2D = sprite.createGraphics(); // Context for buffered image
  // plus the rest of the method...
}
```

sprite オブジェクトの幅と高さはspriteSize で、イメージの型はTYPE_INT_ARGB なので、各ピクセルのアルファ・コンポーネントとカラー・コンポーネントは、単一のint 値として格納され、カラーは8ビットの赤、緑、青のコンポーネントとして格納されます。つまり、sprite イメージは40,000バイトを占有します。これは、Webページのブラウズで消費されるメモリー容量を示すものに過ぎません。このメモリーは、アプレットが実行されるときにローカル・マシンで割り振られるので、ページのダウンロード時間には影響しません。ページであるHTMLファイルのコンテンツのほかに、ダウンロード時間は、アプレットの.class ファイルのサイズと、アプレットを実行したときにダウンロードされるイメージやその他のファイルの影響を受けます。

透過な背景の作成

背景は完全に透過にしたいので、sprite イメージではアルファ・チャネルが重要になります。描画する際は、100x100の矩形イメージ全体ではなく、sprite オブジェクトだけを表示します。このためには、sprite イメージ領域全体を透過 (アルファを1.0f) にしてから (つまり、アルファを0.0fにしてから)、上に重ねたいものを不透過として描画します。次に、イメージ全体を透過にするコードを示します。

```
// Clear image with transparent alpha by drawing a rectangle
g2D.setComposite(AlphaComposite.getInstance(AlphaComposite.CLEAR, 0.0f));
Rectangle2D.Double rect = new Rectangle2D.Double(0,0,spriteSize,spriteSize);
g2D.fill(rect);
```

まず、CLEAR 規則に従ってカラー・コンポーネントをゼロに設定し、アルファを0.0fにしてイメージを透過にすることで、AlphaComposite オブジェクトを使ってアルファ・コンポジットを設定します。次に、イメージ全体を覆う矩形を埋め込みます。CLEAR 規則により各ピクセルのフォアグラウンドと背景の小数部はゼロなので、どちらも生成されるピクセルには関係しないため、カラーを設定する必要はありません。ただし、これによって、影響を受けるイメージ・ピクセルが決まるため 矩形の埋め込みは必要です。

ここで、ちょっと回り道をして、イメージの品質に与える影響について説明しましょう。

レンダリングのヒント

レンダリング操作では、多くの面で、品質を取るか、速度を取るかという選択が迫られます。レンダリング操作とは、いわば、処理時間を犠牲にして品質を優先する、ということです。

レンダリング操作が選択できる場合には、必ずデフォルト設定があります。デフォルトはプラットフォームによって異なりますが、レンダリングを実行するGraphics2D オブジェクトでsetRenderingHint() メソッドを呼び出すことで、ユーザー独自の選択を行うこともできます。ただし、これらは単なるヒントであり、コンピューターが、指定したヒントに従ったレンダリング操作のオプションをサポートしていない場合は、そのヒントは何の効果もありません。

次のcreateSprite() メソッド呼び出しを追加することで、アルファ・コンポジット操作から最高の結果を得ることができます。

```
BufferedImage createSprite(int spriteSize)
{
  // Create image with RGB and alpha channel
  BufferedImage sprite = new BufferedImage(spriteSize, spriteSize,
BufferedImage.TYPE_INT_ARGB);
  Graphics2D g2D = sprite.createGraphics(); // Context for buffered image
  // Set best alpha interpolation quality
  g2D.setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
  // Clear image with transparent alpha by drawing a rectangle
  g2D.setComposite(AlphaComposite.getInstance(AlphaComposite.CLEAR, 0.0f));
  Rectangle2D.Double rect = new Rectangle2D.Double(0,0,spriteSize,spriteSize); g2D.fill(rect);
  // plus the rest of the method...
}
```

RenderingHints クラスでは、マップ・コレクション内のGraphics2D オブジェクトに格納する、各種のレンダリング・ヒントを定義します。setRenderingHint() メソッドの引き数は、キーと、そのキーに対応する値です。アルファ・コンポジット・ヒントのキーが、コードの1つ目の引き数で、2つ目の引き数がヒントの値です。このヒントには、ほかに、プラットフォーム・デフォルトのVALUE_ALPHA_INTERPOLATION_DEFAULT と、品質よりも速度を優先するVALUE_ALPHA_INTERPOLATION_SPEED という値があります。

また、次のキーに対応するヒントを提供することもできます。

キー	説明
KEY_ANTIALIASING	アンチエイリアシングを実行するかどうかを指定する。ギザギザした線をレンダリングする場合、ピクセルは通常、階段方式で配置されるので、線は滑らかに見えない。これをジャギー と言う場合もある。アンチエイリアシング は、ギザギザした線のピクセルの輝度を設定して、線を滑らかに見せる方法である。したがって、このヒントでは、ギザギザした線をレンダリングする場合にジャギーの削減に時間をかけるかどうかを指定する。有効な値はVALUE_ANTIALIAS_ON#_OFF、_DEFAULT。
KEY_COLOR_RENDERING	カラー・レンダリングの実行方法に影響する。有効な値はVALUE_COLOR_RENDER_SPEED#_QUALITY、_DEFAULT。
KEY_DITHERING	ディザリングの処理方法に影響する。ディザリング は、隣接するピクセルのカラーを設定し、限定カラー・セット内にないカラーをまるでそのカラーがあるように表すことで、幅広い範囲のカラーを合成するプロセスである。有効な値はVALUE_DITHER_ENABLE#_DISABLE、_DEFAULT。
KEY_FRACTIONALMETRICS	表示テキストの品質に影響する。有効な値はVALUE_FRACTIONALMETRICS_ON#_OFF、_DEFAULT。
KEY_INTERPOLATION	補間を実行する方法を指定する。 ソース・イメージを変換した場合、変換後のピクセルが宛先のピクセル位置に正確に対応することはめったにない。この場合、変換後の各ピクセルのカラー値は、周囲のピクセルから求めなければならない。補間 は、これを実行するプロセスである。さまざまな方法を使用することができる。有効な値は、処理時間が長いものから順に、VALUE_INTERPOLATION_BICUBIC#_BILINEAR、_NEAREST_NEIGHBOR。
KEY_RENDERING	レンダリング法、犠牲となる速度、および品質を指定する。有効な値はVALUE_RENDERING_SPEED#_QUALITY、_DEFAULT。
KEY_TEXT_ANTIALIASING	テキストをレンダリングする際にアンチエイリアシングを実行するかどうかを指定する。有効な値はVALUE_TEXT_ANTIALIASING_ON#_OFF、_DEFAULT。

これで、回り道は十分です。`sprite` の描画に戻りましょう。

関連トピック

- Amazon.com でBeginning Java 2 - JDK 1.3 Edition をご購入ください。