Java 2D Graphics, The Composite Interface, GradientPaint, and Transparency

by Richard G. Baldwin
baldwin@austin.cc.tx.us

Java Programming, Lecture Notes # 322

March 20, 2000

Introduction

In an earlier lesson, I explained that the **Graphics2D** class extends the **Graphics** class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. Beginning with JDK 1.2, **Graphics2D** is the fundamental class for rendering two-dimensional shapes, text and images.

You also need to understand some other classes and interfaces

I also explained that without understanding the behavior of other classes and interfaces, it is not possible to fully understand the inner workings of the **Graphics2D** class.

Throughout this series of lessons, I have been providing you with information and sample programs designed to help you understand the various classes and interfaces that are necessary for an understanding of the **Graphics2D** class.

Two ways to achieve transparency

There are at least two different ways to achieve transparency in Java 2D. One approach is to use new constructors for the **Color** class that allow you to create solid colors with a specified degree of transparency. I will discuss that approach in a subsequent lesson.

A more general approach

A second, and possibly more general approach, is to make use of an object that implement the **Composite** interface, passing a reference to that object to the **setComposite()** method of the **Graphics2D** class.

You already know about solid colors

An earlier lesson explained the use of the **Composite** interface for solid colors.

Color gradients

This lesson is designed to give you an understanding of the combination of color gradients and the **Composite** interface.

What is the AlphaComposite Class?

The **setComposite()** method requires a reference to an object that implements the **Composite** interface. There is only one class in JDK 1.2.2 that implements the **Composite** interface: **AlphaComposite**.

Many compositing rules

An object of the **AlphaComposite** class can be used to implement any one of about eight different compositing rules.

When you draw something, a new source pixel can overlay an existing destination pixel. The manner in which the color components of the destination pixel are determined depends on the specific rule being applied.

Flanagan explains the rules

You can read about the different rules in Java Foundation Classes in a Nutshell, by David Flanagan.

In these lessons, I will illustrate only one of the compositing rules: the rule known as **SRC_OVER**.

According to Flanagan, this is “By far the most commonly used compositing rule.” I explained this rule in detail in an earlier lesson, so I won’t repeat that explanation here.

How Do I Get an AlphaComposite Object?

You cannot directly instantiate an object of the **AlphaComposite** class. Rather, you get an **AlphaComposite** object by invoking the following factory method of the **AlphaComposite** class.

public static AlphaComposite **getInstance**(int rule, float alpha)

Creates an **AlphaComposite** object with the specified rule and the constant alpha to multiply with the alpha of the source. The source is multiplied with the specified alpha before being composited with the destination.

Parameters:

- rule - the compositing rule
- alpha - the constant alpha to be multiplied with the alpha of the source. alpha must be a floating point number in the inclusive range [0.0, 1.0].

You specify the rule by passing an **int** value given by one of the symbolic constants of the **AlphaComposite** class, such as **SRC_OVER** described earlier.

What is the setComposite() Method?

Here is part of what Sun has to say about the **setComposite()** method of the **Graphics2D** class.

public abstract void **setComposite**(
    Composite comp)

Sets the **Composite** for the **Graphics2D** context. The **Composite** is used in all drawing methods such as drawImage, drawString, draw, and fill. It specifies how new pixels are to be combined with the existing pixels on the graphics device during the rendering process.

Parameters:

- comp - the **Composite** object to be used for rendering

The required parameter is a reference to any object that implements the **Composite** interface, meaning that you could define your own class to implement this interface.

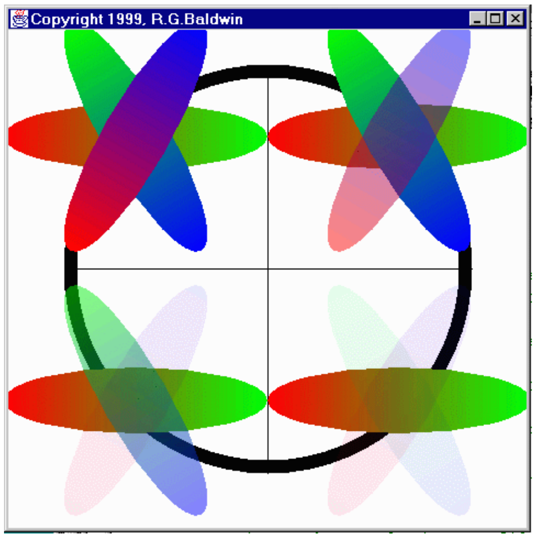
In this lesson, I elected to make use of the existing **AlphaComposite** class described above.

Sample Program

This program is named **Composite02**. You will need to compile and execute the program so that you can view its output while reading the discussion. Without being able to view the output, the discussion will probably mean very little to you.

A screen shot of the output

In case you are unable, for some reason, to compile and run the program, here is a screen shot of the program output. However, this image has been reduced to about seventy-percent of its original size in pixels. Therefore, some of the quality was lost in the reduction process.



The GUI is a Frame object

The program draws a four-inch by four-inch **Frame** on the screen. It translates the origin to the center of the **Frame**. Then it draws a pair of X and Y-axes centered on the new origin.

A large circle

After drawing the X and Y-axes, the program draws a circle with a thick border centered on the origin. This circle is used later to provide visual cues relative to transparency.

Transparent ellipses

After the large circle is drawn, three ellipses are drawn on top of one another in each quadrant.

Each ellipse has a common center, and is rotated by sixty degrees relative to the ellipse beneath it.

Color gradient

The color of each ellipse is based on a color gradient. The color of the ellipse on the bottom of the stack is a gradient from red to green. The ellipse in the center of the stack is a gradient from green to blue. The ellipse on the top of the stack is a gradient from blue to red.

Different transparency values

The different ellipses are given various transparency values in the different quadrants to illustrate the effect of the alpha parameter of the **setComposite()** method.

Transparency by quadrant

Here is the transparency given to each of the ellipses in the different quadrants.

TRANSPARENCY
Upper-left quadrant
No transparency

Upper-right quadrant
Red to green is not transparent
Green to blue is not transparent
Blue to red is 50-percent transparent

Lower-left quadrant
Red to green is not transparent
Green to blue is 50-percent transparent
Blue to red is 90-percent transparent

Lower-right quadrant
Red to green is not transparent
Green to blue is 90-percent transparent
Blue to red is 90-percent transparent

As you can see from the information given above, the red-to-green ellipse is opaque in all four quadrants. As a result, the large black circle doesn't show through the red-to-green ellipse in any of the quadrants.

Upper-left quadrant

All three ellipses are opaque in the upper-left quadrant, so nothing shows through, and the stacking order of the ellipses is pretty obvious.

Other ellipses become transparent

The other two ellipses are made progressively more transparent as you move through the other three quadrants. As a result, you can "see through" the green-to-blue and blue-to-red ellipses. In other words, you can see the geometric figures that lie beneath them (the other ellipses and the large black circle).

Upper-right quadrant

In this quadrant, the blue-to-red ellipse is transparent, but the green-to-blue ellipse and the red-to-green ellipse are opaque. Neither the large circle nor the red-to-green ellipse can be seen through the green-to-blue ellipse.

However, both of the other ellipses show through the blue-to-red ellipse in the upper-right quadrant.

Similar to previous lesson

The material in this lesson is very similar to a previous lesson except for the use of color gradients in this lesson as opposed to solid colors in the previous lesson. Therefore, I am not going to discuss the output of this program in detail.

Illustrates rotation and translation

As mentioned in the earlier lesson, this lesson also provides a good illustration of the benefits of rotation and translation.

Uses the AffineTransform

The task of rotating the ellipses relative to each other and the task of translating them into the various quadrants was made much easier (even possible) through the use of the **AffineTransform** to rotate and translate the ellipses.

Rotation was especially useful in this lesson

The benefits of rotation are particularly significant in this lesson where it was necessary to specify the coordinates of the ends of each ellipse to create the color gradient. Because this was accomplished before rotating the ellipses, the task was very easy. Had it been necessary to specify the ends of rotated ellipses, the task would have been much more difficult.

The normal caveat regarding inches

This discussion of dimensions in inches on the screen depends on the method named **getScreenResolution()** returning the correct value. However, the **getScreenResolution()** method always seems to return 120 on my computer regardless of the actual screen resolution settings.

Will discuss in fragments

I will briefly discuss this program in fragments. The controlling class and the constructor for the GUI class are essentially the same as you have seen in several previous lessons, so, I won't repeat that discussion here. You can view that material in the complete listing of the program at the end of the lesson.

All of the interesting action takes place in the overridden **paint()** method, so I will begin the discussion there.

Overridden paint() method

The beginning portions of the overridden **paint()** method should be familiar to you by now as well. So, I am going to let the comments in Figure 1 speak for themselves.

```
publicvoid paint(Graphics g){
  //Downcast the Graphics object to a
  // Graphics2D object
  Graphics2D g2 = (Graphics2D)g;

  //Scale device space to produce inches on
  // the screen  based on actual screen
  // resolution.
  g2.scale((double)res/72,(double)res/72);

  //Translate origin to center of Frame
  g2.translate((hSize/2)*ds,(vSize/2)*ds);

  //Draw x-axis
  g2.draw(new Line2D.Double(
                    -1.5*ds,0.0,1.5*ds,0.0));
  //Draw y-axis
  g2.draw(new Line2D.Double(
                    0.0,-1.5*ds,0.0,1.5*ds));
```

Figure 1

The large circle

The code in Figure 2 draws the large circle with a border width of 0.1 inches. There is nothing new here, so I won't provide an explanation.

```
//Draw a big circle underneath all of the
// ellipses
```

```
g2.setStroke(new BasicStroke(0.1f*ds));
Ellipse2D.Double bigCircle =
            new Ellipse2D.Double(
             -1.5*ds,-1.5*ds,3.0*ds,3.0*ds);
g2.draw(bigCircle);
```

Figure 2

An ellipse reference variable

Figure 3 simply declares a reference variable of the class **Ellipse2D.Double**. This reference variable will be used repeatedly in subsequent code for the instantiation of ellipse objects.

```
  Ellipse2D.Double theEllipse;
```

Figure 3

Translation

The code in Figure 4 translates the origin to the center of what was previously the upper-left quadrant. After this statement is executed, any geometric figure that is drawn centered on the origin will actually be rendered in the center of what was earlier the upper-left quadrant.

```
   g2.translate(-1.0*ds,-1.0*ds);
```

Figure 4

Opaque red-to-green ellipse

The code in Figure 5 draws and fills an opaque ellipse with a red-to-green gradient centered on the new origin.

```
   theEllipse = new Ellipse2D.Double(
             -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
   g2.setPaint(new GradientPaint(
                -1.0f*ds,0.0f*ds,Color.red,
                 1.0f*ds,0.0f*ds,Color.green));
   //Red to green is not transparent
   g2.setComposite(
        AlphaComposite.getInstance(
             AlphaComposite.SRC_OVER,1.0f));
   g2.fill(theEllipse);
```

Figure 5

The code has been covered in previous lessons, so I won't discuss it further in this lesson. The only thing new here is the combined use of **GradientPaint** and **setComposite()**.

From this point on, everything is pretty much business as usual, so I will terminate the discussion at this point and refer you to the complete listing of the program at the end of the lesson.

Summary

In this lesson, I have shown you how to use the **setComposite()** method of the **Grapics2D** class along with the **AlphaComposite** class to control the manner in which new pixel values are composited with existing pixel values.

The new material in the lesson was the combined use of **GradientPaint** and **setComposite()**.

The sample program in this lesson concentrates on transparency as a way to demonstrate compositing pixels.

In addition, you have seen some additional uses for the **translate** and **rotate** transforms.

Complete Program Listing

A complete listing of the program is provided in Figure 6.

```
/*Composite02.java 12/12/99
 Copyright 1999, R.G.Baldwin

 Identical to Composite01 except that it uses gradient
 colors instead of solid colors.

 Illustrates use of the AlphaComposite class to
 achieve transparency with gradient-fill colors.

 Draws a 4-inch by 4-inch Frame on the screen.

 Translates the orgin to the center of the Frame.

 Draws a pair of X and Y-axes centered on the new
 origin.

 Draw a big circle centered on the origin underneath
 all of the ellipses.

 Uses rotation and translation to fill three ellipses in
 each of the four quadrants.  The ellipses intersect at
 their center.  Each is rotated by 60 degrees relative
 to the one below it.  The order is:
   Red to green gradient ellipse on the bottom
   Green to blue gradient ellipse in the middle
   Blue to red gradient ellipse on the top

 TRANSPARENCY
 Upper-left quadrant
 No transparency

 Upper-right quadrant
 Red to green is not transparent
 Green to blue is not transparent
 Blue to red is 50-percent transparent

 Lower-left quadrant
 Red to green is not transparent
```

```
Green to blue is 50-percent transparent
Blue to red is 90-percent transparent

Lower-right quadrant
Red to green is not transparent
Green to blue is 90-percent transparent
Blue to red is 90-percent transparent


Whether the dimensions in inches come out right or
not depends on whether the method
getScreenResolution() returns the correct resolution
for your screen.

Tested using JDK 1.2.2 under WinNT Workstation 4.0
****************************************/
import java.awt.geom.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;

class Composite02{
  publicstaticvoid main(String[] args){
    GUI guiObj = new GUI();
  }//end main
}//end controlling class Composite02

class GUI extends Frame{
  int res;//store screen resolution here
  staticfinalint ds = 72;//default scale, 72 units/inch
  staticfinalint hSize = 4;//horizonal size = 4 inches
  staticfinalint vSize = 4;//vertical size = 4 inches

  GUI(){//constructor
    //Get screen resolution
    res = Toolkit.getDefaultToolkit().
                                getScreenResolution();
    //Set Frame size
    this.setSize(hSize*res,vSize*res);
    this.setVisible(true);
    this.setTitle("Copyright 1999, R.G.Baldwin");

    //Window listener to terminate program.
    this.addWindowListener(new WindowAdapter(){
      publicvoid windowClosing(WindowEvent e){
        System.exit(0);}});
  }//end constructor
  //----------------------------------------------------//

  //Override the paint() method
  publicvoid paint(Graphics g){
    //Downcast the Graphics object to a
    // Graphics2D object
    Graphics2D g2 = (Graphics2D)g;

    //Scale device space to produce inches on the
    // screen based on actual screen resolution.
    g2.scale((double)res/72,(double)res/72);

    //Translate origin to center of Frame
    g2.translate((hSize/2)*ds,(vSize/2)*ds);

    //Draw x-axis
    g2.draw(new Line2D.Double(
                                -1.5*ds,0.0,1.5*ds,0.0));
    //Draw y-axis
    g2.draw(new Line2D.Double(
                                0.0,-1.5*ds,0.0,1.5*ds));

    //Draw a big circle underneath all of the ellipses.
```

```
g2.setStroke(new BasicStroke(0.1f*ds));
Ellipse2D.Double bigCircle =
                  new Ellipse2D.Double(
                       -1.5*ds,-1.5*ds,3.0*ds,3.0*ds);
g2.draw(bigCircle);

Ellipse2D.Double theEllipse;


//Translate origin to upper-left quadrant
g2.translate(-1.0*ds,-1.0*ds);

//Red to green horizontal ellipse
theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.setPaint(new GradientPaint(
                       -1.0f*ds,0.0f*ds,Color.red,
                        1.0f*ds,0.0f*ds,Color.green));
//Red to green is not transparent
g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
g2.fill(theEllipse);

//Green to blue ellipse at 60 degrees
theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate(Math.PI/3.0);//rotate 60 degrees
g2.setPaint(new GradientPaint(
                        -1.0f*ds,0.0f*ds,Color.green,
                         1.0f*ds,0.0f*ds,Color.blue));
//Green to blue is not transparent
g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
g2.fill(theEllipse);

//Blue to red ellipse at 120 degrees
theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate((Math.PI/3.0));//rotate 60 more degrees
g2.setPaint(new GradientPaint(
                         -1.0f*ds,0.0f*ds,Color.blue,
                          1.0f*ds,0.0f*ds,Color.red));
//Blue to red is not transparent
g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
g2.fill(theEllipse);


//Translate origin to upper-right quadrant
g2.rotate(-2*(Math.PI/3.0));//undo prev rotation
g2.translate(2.0*ds,0.0*ds);

//Red to green horizontal ellipse
theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.setPaint(new GradientPaint(
                       -1.0f*ds,0.0f*ds,Color.red,
                        1.0f*ds,0.0f*ds,Color.green));
//Red to green is not transparent
g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
g2.fill(theEllipse);

//Green to blue ellipse at 60 degrees
theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
g2.rotate(Math.PI/3.0);//rotate 60 degrees
g2.setPaint(new GradientPaint(
                        -1.0f*ds,0.0f*ds,Color.green,
                         1.0f*ds,0.0f*ds,Color.blue));
```

```
    //Green to blue is not transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
    g2.fill(theEllipse);

    //Blue to red ellipse at 120 degrees
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.rotate((Math.PI/3.0));//rotate 60 more degrees
    g2.setPaint(new GradientPaint(
                         -1.0f*ds,0.0f*ds,Color.blue,
                          1.0f*ds,0.0f*ds,Color.red));
    //Blue to red is 50-percent transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,0.5f));
    g2.fill(theEllipse);


    //Translate origin to lower-left quadrant
    g2.rotate(-2*(Math.PI/3.0));//undo prev rotation
    g2.translate(-2.0*ds,2.0*ds);

    //Red to green horizontal ellipse
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.setPaint(new GradientPaint(
                       -1.0f*ds,0.0f*ds,Color.red,
                        1.0f*ds,0.0f*ds,Color.green));
    //Red to green is not transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
    g2.fill(theEllipse);

    //Green to blue ellipse at 60 degrees
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.rotate(Math.PI/3.0);//rotate 60 degrees
    g2.setPaint(new GradientPaint(
                        -1.0f*ds,0.0f*ds,Color.green,
                         1.0f*ds,0.0f*ds,Color.blue));
    //Green to blue is 50 percent transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,0.5f));
    g2.fill(theEllipse);

    //Blue to red ellipse at 120 degrees
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.rotate((Math.PI/3.0));//rotate 60 more degrees
    g2.setPaint(new GradientPaint(
                         -1.0f*ds,0.0f*ds,Color.blue,
                          1.0f*ds,0.0f*ds,Color.red));
    //Blue to red is 90-percent transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,0.1f));
    g2.fill(theEllipse);


    //Translate origin to lower-right quadrant
    g2.rotate(-2*(Math.PI/3.0));//undo prev rotation
    g2.translate(2.0*ds,0.0*ds);

    //Red to green horizontal ellipse
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.setPaint(new GradientPaint(
                       -1.0f*ds,0.0f*ds,Color.red,
                        1.0f*ds,0.0f*ds,Color.green));
    //Red to green is not transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,1.0f));
    g2.fill(theEllipse);

    //Green to blue ellipse at 60 degrees
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.rotate(Math.PI/3.0);//rotate 60 degrees
    g2.setPaint(new GradientPaint(
                        -1.0f*ds,0.0f*ds,Color.green,
                         1.0f*ds,0.0f*ds,Color.blue));
    //Green to blue is 90-percent transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,0.1f));
    g2.fill(theEllipse);

    //Blue to red ellipse at 120 degrees
    theEllipse = new Ellipse2D.Double(
                      -1.0*ds,-0.25*ds,2.0*ds,0.5*ds);
    g2.rotate((Math.PI/3.0));//rotate 60 more degrees
    g2.setPaint(new GradientPaint(
                         -1.0f*ds,0.0f*ds,Color.blue,
                          1.0f*ds,0.0f*ds,Color.red));
    //Blue to red is 90-percent transparent
    g2.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.SRC_OVER,0.1f));
    g2.fill(theEllipse);

  }//end overridden paint()

}//end class GUI
//===============================//
```

Figure 6

About the author

Richard Baldwin *is a college professor and private consultant whose primary focus is a combination of Java and XML. In addition to the many platform-independent benefits of Java applications, he believes that a combination of Java and XML will become the primary driving force in the delivery of structured information on the Web.*

Richard has participated in numerous consulting projects involving Java, XML, or a combination of the two. He frequently provides onsite Java and/or XML training at the high-tech companies located in and around Austin, Texas. He is the author of Baldwin's Java Programming Tutorials, which has gained a worldwide following among experienced and aspiring Java programmers. He has also published articles on Java Programming in Java Pro magazine.

Richard holds an MSEE degree from Southern Methodist University and has many years of experience in the application of computer technology to real-world problems.

-end-