Java 2D Graphics, Gradient Color Fill

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

Java Programming, Lecture Notes # 314

March 19, 2000

- Introduction
- The Three Paint Classes
- Sample Program
- Summary
- Complete Program Listings

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.

Understanding other classes also required

I also explained that without understanding the behavior of other classes and interfaces such as **Shape**, **AffineTransform**, **GraphicsConfiguration**, **PathIterator**, and **Stroke**, it is not possible to fully understand the inner workings of the **Graphics2D** class.

What has been covered previously?

Earlier lessons have explained a number of Java 2D concepts, including **Shape**, **AffineTransform**, and **PathIterator**.

Before, I can explain the **Stroke** class, I need to explain how to fill a **Shape** in general. An earlier lesson showed you how to fill a **Shape** with a solid color.

How to fill, in general

I explained in an earlier lesson that if you want to *fill* a **Shape** object before you draw it, you can accomplish this with the following two steps:

- Invoke **setPaint()** on the **Graphics2D** object, passing a reference to an object of a class that implements the **Paint** interface as a parameter.
- Invoke the **fill()** method on the **Graphics2D** object, passing a reference to the **Shape** object that you want to fill as a parameter.

Filling with color gradient

This lesson will show you how to fill a **Shape** with a *color gradient*, both *cyclic* and *acyclic*.

The Three Paint Classes

In a previous lesson, I explained that the Java2D API in JDK 1.2.2 provides three classes that implement the **Paint** interface (and you can also define your own):

- **Color**
- **GradientPaint**
- **TexturePaint**

The Color class

The **Color** class can be used to fill a **Shape** object with a solid color. That was the topic of an earlier lesson.

The GradientPaint class

The **GradientPaint** class can be used to fill a **Shape** with a color gradient. The gradient progresses from one specified color at one point in user space to a different specified color at a different point in user space.

An *acyclic* gradient

The two points describe a hypothetical line segment in user space. The two colors can be stabilized beyond the end points of the hypothetical line segment. This is known as an *acyclic* gradient.

A *cyclic* gradient

The gradient can also be caused to repeat in a *cyclic* fashion beyond the end points of the hypothetical line segment. This is known as a *cyclic* gradient.

The use of the **GradientPaint** class is the primary topic of this lesson.

The TexturePaint class

The **TexturePaint** class can be used to fill a **Shape** with a tiled version of a **BufferedImage** object. This will also be the topic of a subsequent lesson.

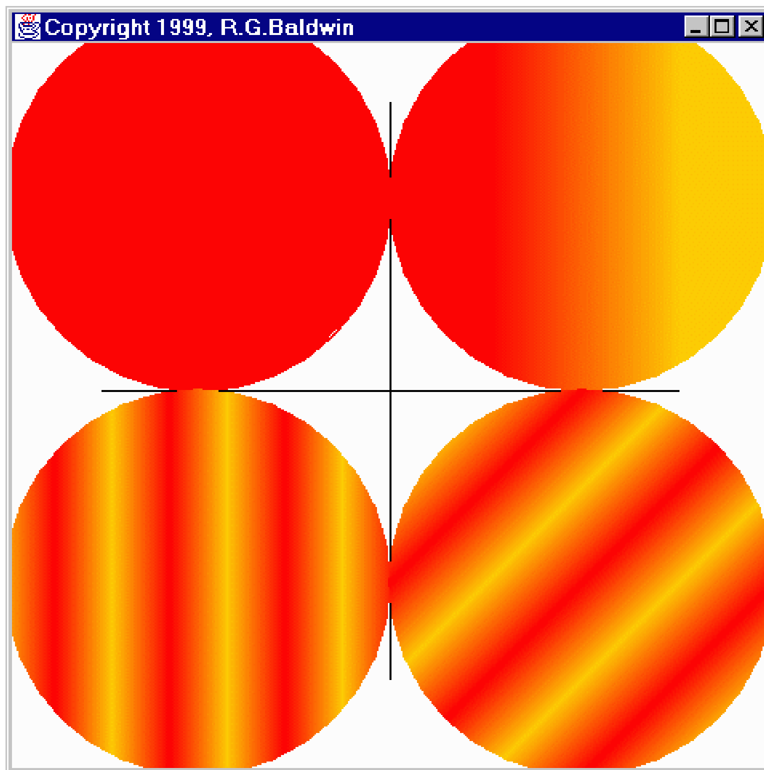
Sample Program

The name of this program is **PaintGradient01**. It illustrates the use of a **Paint** object to fill a Shape with a solid color.

In this case, the **Paint** object is an instance of the **GradientPaint** class, which implements the interface named **Paint**.

A screen shot of the output

A significantly reduced screen shot of the output of this program is shown below. Note that this screen shot was reduced to about seventy-percent of its original size in pixels. Thus some of the quality was lost in the 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.

So far, this is very similar to the sample programs that I have explained in previous lessons.

A circle in each quadrant

The program then draws one two-inch diameter circle in each quadrant. For purpose of reference, it fills the circle in the upper left quadrant with solid red, exactly as in an earlier program.

The circles in the other three quadrants are filled with color gradients that progress from red to orange in different ways.

Acyclic gradient on the horizontal

The color gradient in the upper right-hand circle progresses from red on the left end of a hypothetical line to orange on the right end of a hypothetical line in an *acyclic* manner.

In other words, everything to the left of the beginning of a hypothetical line is the same color red. Everything to the right of the end of the hypothetical line is the same color orange. Only that portion in between the beginning and the end points of the hypothetical line vary in color.

Cyclic gradient on the horizontal

The color gradient in the lower left-hand circle progresses from red to orange and back several times in a *cyclic* manner. The variations in color progress along and beyond a hypothetical line that is parallel to the horizontal axis.

Cyclic gradient at 45 degrees to the horizontal

The color gradient in the lower right-hand circle also progresses from red to orange and back several times in a *cyclic* manner. However, in this case, the variations in color progress along and beyond a hypothetical line that is angled at 45 degrees to the horizontal.

The normal disclaimer on inches

The program was tested using JDK 1.2.2 under WinNT Workstation 4.0

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

As is often the case, I will 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 bore you by repeating that discussion here. You can view that material in the complete listing of the program near 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.

```
public void 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));

  //Upper left quadrant, Solid red fill
  Ellipse2D.Double circle1 =
     new Ellipse2D.Double(
             -2.0*ds,-2.0*ds,2.0*ds,2.0*ds);
  g2.setPaint(new Color(255,0,0));//red
  g2.fill(circle1);
  g2.draw(circle1);
```

Figure 1

The code in Figure 1 includes the code required to place the circle in the upper left quadrant and fill it with the solid color red. This is the same code that I showed you in an earlier lesson on solid-color fill.

The interesting part

That brings us to the interesting part, which is to place a circle in the upper-right quadrant and fill it with a horizontal, *acyclic* gradient from red to orange.

I begin by instantiating an object of the **Ellipse2D.Double** class bounded by a square in the upper-right quadrant. This is a circle.

This is not new. You have seen code like this in previous lessons, so I won't discuss it further. See Figure 2.

```
    Ellipse2D.Double circle2 =
            new Ellipse2D.Double(
                0.0*ds,-2.0*ds,2.0*ds,2.0*ds);
```

Figure 2

The GradientPaint class

At this point, we need to take a look at some detailed information about the **GradientPaint** class. This is what Sun has to say on the topic.

"The **GradientPaint** class provides a way to fill a **Shape** with a linear color gradient pattern.

If Point P1 with Color C1 and Point P2 with Color C2 are specified in user space, the Color on the P1, P2 connecting line is proportionally changed from C1 to C2.

Any point P not on the extended P1, P2 connecting line has the color of the point P' that is the perpendicular projection of P on the extended P1, P2 connecting line.

Points on the extended line outside of the P1, P2 segment can be colored in one of two ways.

- If the gradient is *cyclic* then the points on the extended P1, P2 connecting line cycle back and forth between the colors C1 and

C2.

- If the gradient is *acyclic* then points on the P1 side of the segment have the constant Color C1 while points on the P2 side have the constant Color C2."

For the record, the gradient implemented by the next code fragment is *acyclic*.

GradientPaint constructor

Now, we need to take a look at the constructor for the **GradientPaint** class.

Four overloaded versions

Actually, there are four overloaded versions of the constructor. Two of them accept the coordinates of the ends of the hypothetical line mentioned above (P1 and P2) as objects of the class **Point2D** (I discussed the **Point2D** class in one of the early lessons in this series on Java 2D).

The other two constructors accept the coordinates of the ends of the hypothetical line as parameters of type **float**. (If you need the accuracy of **double**, you need to use one of the constructors that accept objects of the class **Point2D**.)

The constructor in the next code fragment specifies the end points of the hypothetical line as type **float**.

Cyclic* versus *acyclic

Having separated the constructors into two categories based on how the coordinate information is specified, the next separation has to do with *acyclic* versus *cyclic* behavior.

Two of the constructors (one of each coordinate data type) default to *acyclic* behavior.

The other two constructors have a **boolean** parameter that allows you to specify *cyclic* or *acyclic*.

One specific constructor

Here is information about the version of the constructor that accepts **float** parameters and allows you to specify *acyclic* or *cyclic*.

public **GradientPaint**(
 float x1,

float y1,
Color color1,
float x2,
float y2,
Color color2,
boolean cyclic)

Constructs either a *cyclic* or *acyclic* **GradientPaint** object depending on the boolean parameter.

Parameters:

- x1, y1 - coordinates of the first specified Point in user space
- color1 - Color at the first specified Point
- x2, y2 - coordinates of the second specified Point in user space
- color2 - Color at the second specified Point
- cyclic - true if the gradient pattern should cycle repeatedly between the two colors; false otherwise

This is the version of the constructor that is used in the next code fragment.

Where's the code?

The next fragment (Figure 3) invokes the **setPaint()** method passing a reference to an object that implements the **Paint** interface as a parameter. As you are already aware, the parameter to **setPaint()** must implement the **Paint** interface.

```
  g2.setPaint(
    new GradientPaint(
      0.5f*ds,-1.0f*ds,Color.red,
      1.5f*ds,-1.0f*ds,Color.orange,false));
```

Figure 3

A GradientPaint object

The thing that is new about this fragment is that the object that is passed to the **setPaint()** method is an object of the **GradientPaint** class. (Did I mention that **GradientPaint** also implements **Paint**?)

This **GradientPaint** object will be used to fill a circle that is two inches in diameter. The bounding rectangle for the circle is a square that fits exactly in the upper-right quadrant of the **Frame.**

End points of the hypothetical line

The hypothetical line segment that determines the beginning and the end of the color gradient begins at a point that is one-half inch inside the left edge of the circle (**0.5f*ds**). The hypothetical line segment stops at a point that is one-half inch inside the right edge of the circle (**1.5f*ds**).

The hypothetical line segment is horizontal

The coordinate information describes the ends of a hypothetical line that is parallel to the horizontal axis (the Y-coordinates of the two end points of the hypothetical line are the same at **-1.0f*ds**).

float rather than double

In case this syntax is new to you, the "f" causes the literal value to be interpreted as **float** rather than **double**.

Color gradient is horizontal

Since the hypothetical line is parallel to the horizontal axis, the color gradient will also be parallel to the horizontal axis.

Color gradient is *acyclic*

The **boolean** parameter is false, so the gradient does not repeat beyond the ends of the hypothetical line.

Gradient is from red to orange

The beginning color is red, so everything to the left of the starting point is red.

The ending color is orange, so everything to the right of the ending point is orange.

In between, the color varies from red to orange.

Fill the circle and render it

As in an earlier lesson, Figure 4 invokes the **fill()** method to fill the circle with the color gradient, and then invokes the **draw()** method to render the circle on the screen. There is nothing new here.

```
  g2.fill(circle2);
  g2.draw(circle2);
```

Figure 4

Run the program

Run the program and take a look at the circle in the upper-right quadrant.

Someone once said that a picture is worth a thousand explanations of code fragments, or something to that effect.

Looks kind of like the sun

On my machine, the circle looks a little like a photograph of the sun (no reference intended to the company named that invented Java).

Horizontal *cyclic* color gradient

Figure 5 causes a *cyclic* color gradient to be applied to a circle in the lower-left quadrant.

```
 Ellipse2D.Double circle3 =
     new Ellipse2D.Double(
               -2.0*ds,0.0*ds,2.0*ds,2.0*ds);
  g2.setPaint(
      new GradientPaint(
         -1.15f*ds,1.0f*ds,Color.red,
         -0.85f*ds,1.0f*ds,Color.orange,true));
  g2.fill(circle3);
  g2.draw(circle3);
```

Figure 5

The coordinate values that are used cause the circle to be in the lower-left quadrant, and cause the gradient to be parallel to the horizontal axis.

The **true** parameter that is passed to the constructor for the **GradientPaint** object causes the gradient to be *cyclic*.

A *cyclic* gradient at 45 degrees

The final fragment, Figure 6, causes a *cyclic* gradient from red to orange to fill a circle in the lower-right quadrant. It will be left as an exercise for the student to interpret the coordinate values of the end points of the hypothetical line to understand how it represents a line at 45 degrees to the horizontal.

```
  Ellipse2D.Double circle4 =
     new Ellipse2D.Double(
              0.0*ds,0.0*ds,2.0*ds,2.0*ds);
  g2.setPaint(
     new GradientPaint(
       0.0f*ds,0.0f*ds,Color.red,
       0.25f*ds,0.25f*ds,Color.orange,true));
  g2.fill(circle4);
  g2.draw(circle4);
```

Figure 6

Again, the **boolean** value passed to the **GradientPaint** constructor is **true**, causing the gradient to be *cyclic*.

You can view a complete listing of the program at the end of the lesson.

Summary

In this lesson, I have explained the **GradientPaint** class, and have shown you how it can be used to fill a **Shape** with a color gradient that progresses from one color at a specified point in user space to a different color at a different point in user space.

The gradient can be either *cyclic* or *acyclic*, and it can progress along a hypothetical line of any length, at any angle in user space.

Complete Program Listing

A complete listing of the program is provided in Figure 7.

```
/*PaintGradient01.java 12/12/99
 Copyright 1999, R.G.Baldwin

 Illustrates use of a Paint object to fill a Shape with
 a gradient and a solid color.

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

 Translates the origin to the center of the Frame.

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

 Draws one 2-inch diameter circle in each quadrant.

 Fill upper left circle with solid red.
 Fills upper right circle with gradient red to orange,
 acyclic
```

```
Fills lower left circle with gradient red to orange,
cyclic along horizontal axis
Fills lower right circle with gradient, red to orange,
cyclic along 45 degrees

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 PaintGradient01{
  publicstaticvoid main(String[] args){
    GUI guiObj = new GUI();
  }//end main
}//end controlling class PaintGradient01

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(){
      public void windowClosing(WindowEvent e){
        System.exit(0);}});
  }//end constructor
  //----------------------------------------------------//

  //Override the paint() method
  public void 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));

    //Upper left quadrant, Solid red fill
    Ellipse2D.Double circle1 =
       new Ellipse2D.Double(
                     -2.0*ds,-2.0*ds,2.0*ds,2.0*ds);
```

```
    g2.setPaint(new Color(255,0,0));//red
    g2.fill(circle1);
    g2.draw(circle1);

    //Upper right quadrant
    //Gradient red to orange, acyclic
    Ellipse2D.Double circle2 =
       new Ellipse2D.Double(
                        0.0*ds,-2.0*ds,2.0*ds,2.0*ds);
    g2.setPaint(
          new GradientPaint(
                0.5f*ds,-1.0f*ds,Color.red,
                1.5f*ds,-1.0f*ds,Color.orange,false));
    g2.fill(circle2);
    g2.draw(circle2);

    //Lower left quadrant
    //Gradient red to orange, cyclic along
    //horizontal axis
    Ellipse2D.Double circle3 =
       new Ellipse2D.Double(
                       -2.0*ds,0.0*ds,2.0*ds,2.0*ds);
    g2.setPaint(
          new GradientPaint(
                -1.15f*ds,1.0f*ds,Color.red,
                -0.85f*ds,1.0f*ds,Color.orange,true));
    g2.fill(circle3);
    g2.draw(circle3);

    //Lower right quadrant
    //Gradient red to orange, cyclic along
    // 45 degree angle
    Ellipse2D.Double circle4 =
       new Ellipse2D.Double(
                        0.0*ds,0.0*ds,2.0*ds,2.0*ds);
    g2.setPaint(
          new GradientPaint(
                0.0f*ds,0.0f*ds,Color.red,
                0.25f*ds,0.25f*ds,Color.orange,true));
    g2.fill(circle4);
    g2.draw(circle4);

  }//end overridden paint()

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

Figure 7

Copyright 2000, Richard G. Baldwin. Reproduction in whole or in part in any form or medium without express written permission from Richard Baldwin is prohibited.

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-