Magic with Merlin: Porter-Duff rules!

Java 2D adds the remaining four rules

John Zukowski

September 01, 2001

Two-dimensional graphics programming with the Java language just got a little better. The `AlphaComposite` class, which previously supported only eight of the 12 Porter-Duff rules for digital image compositing, now supports all 12. In this installment of *Magic with Merlin*, John Zukowski describes all 12 rules and offers an interactive program to demonstrate how they operate.

View more content in this series

Back in 1984, Thomas Porter and Tom Duff wrote a paper entitled "Compositing Digital Images" that described 12 rules combining two images. Support for these *compositing* rules is found in the `AlphaComposite` class, first introduced in version 1.2 of the Java language. Version 1.4, currently in beta 2, supports all 12 rules.

Support for compositing is necessary for applications like games that include multiple image types, including background images, player-character images, and the like. While it is easy to always draw the player in front of the background, if the player were to jump behind a tree, you would want to show the character image faded out somewhat behind the tree image. This is where compositing comes in handy.

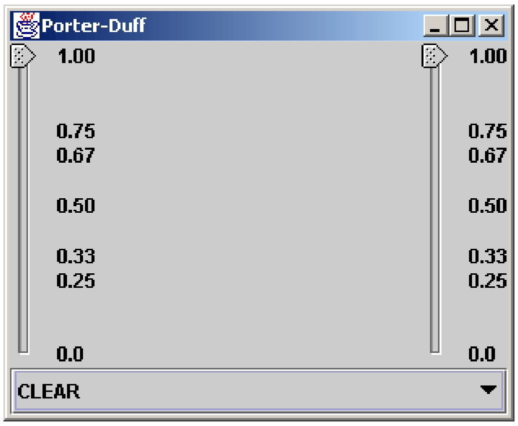
Porter and Duff's 12 rules

The `AlphaComposite` class has 12 constants, one for each rule. It can be tricky to visualize how the rule is applied, so I've provided an illustration of each rule in action. The actual combinations will vary if the images aren't opaque. The demonstration program, shown in Listing 1 and available for download in the Related topics section, allows you to try out different levels of transparency.

Note: Those rules newly added with the Merlin release are indicated with an asterisk (*).

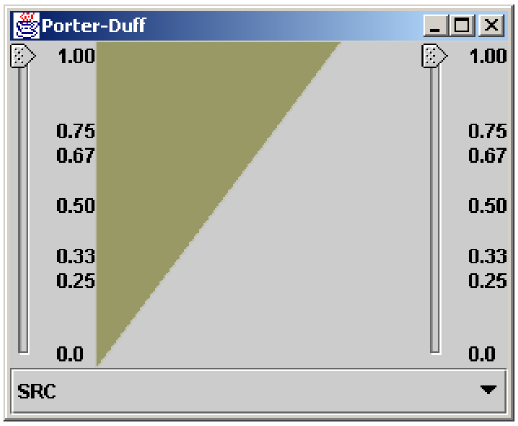
Rule 1.`AlphaComposite.CLEAR`: Nothing will be drawn in the combined image.

AlphaComposite.CLEAR



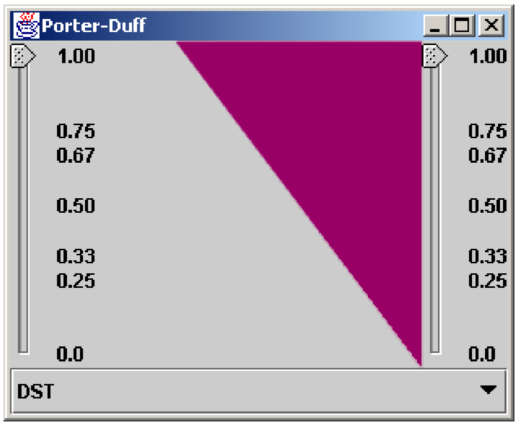
Rule 2.`AlphaComposite.SRC`: SRC stands for source; only the source image will be in the combined image.

AlphaComposite.SRC



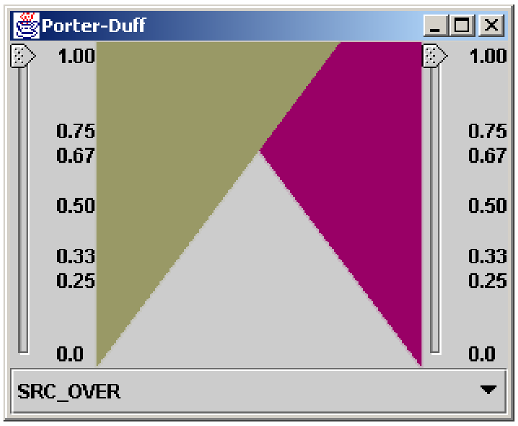
Rule 3.`AlphaComposite.DST`*: DST stands for destination; only the destination image will be in the combined image.

AlphaComposite.DST



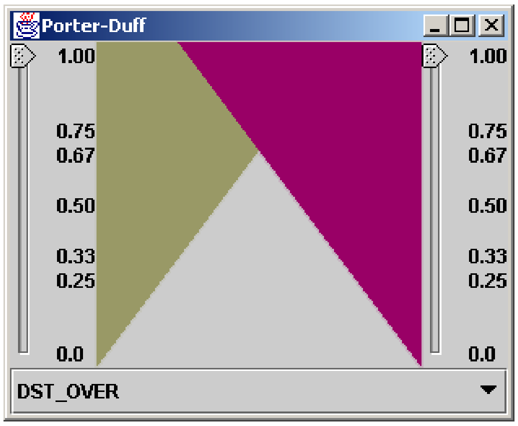
Rule 4.`AlphaComposite.SRC_OVER`: The source image will be drawn over the destination image.

AlphaComposite.SRC_OVER



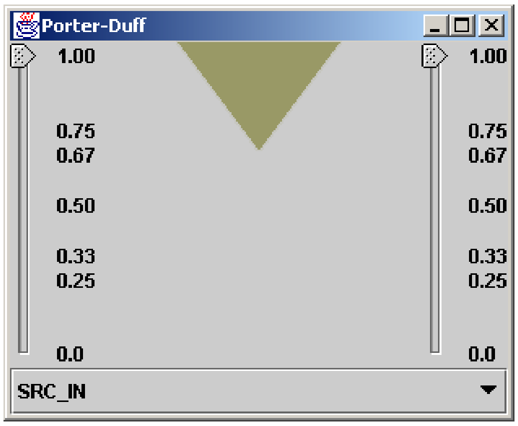
Rule 5.`AlphaComposite.DST_OVER`: The destination image will be drawn over the source image.

AlphaComposite.DST_OVER



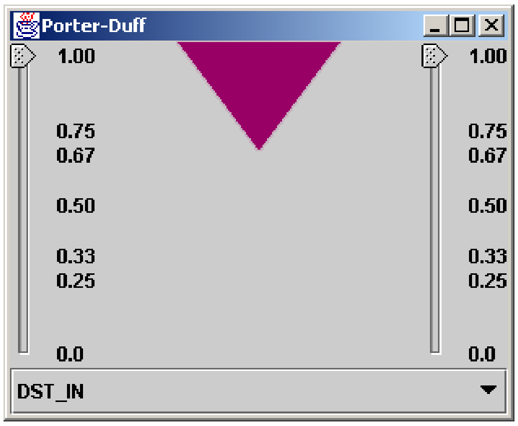
Rule 6.`AlphaComposite.SRC_IN`: Only the part of the source image that overlaps the destination image will be drawn.

AlphaComposite.SRC_IN



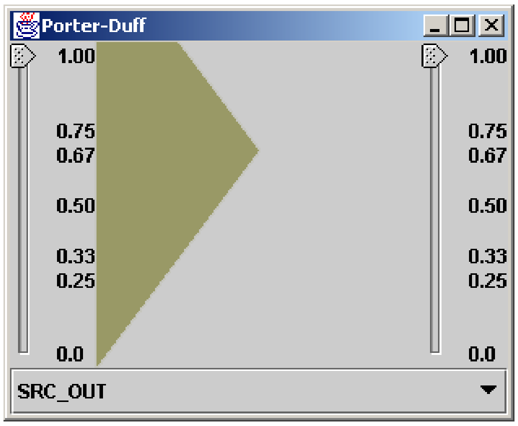
Rule 7.`AlphaComposite.DST_IN`: Only the part of the destination image that overlaps the source image will be drawn.

AlphaComposite.DST_IN



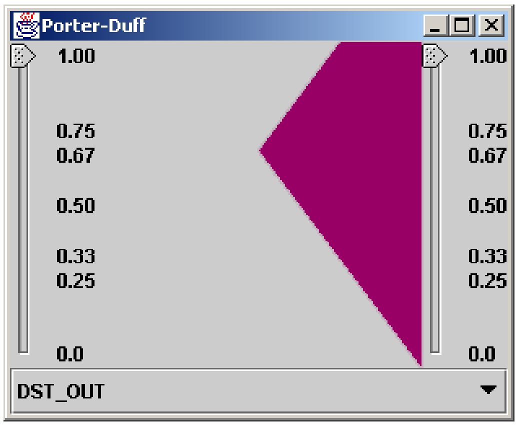
Rule 8.`AlphaComposite.SRC_OUT`: Only the part of the source image that doesn't overlap the destination image will be drawn.

AlphaComposite.SRC_OUT



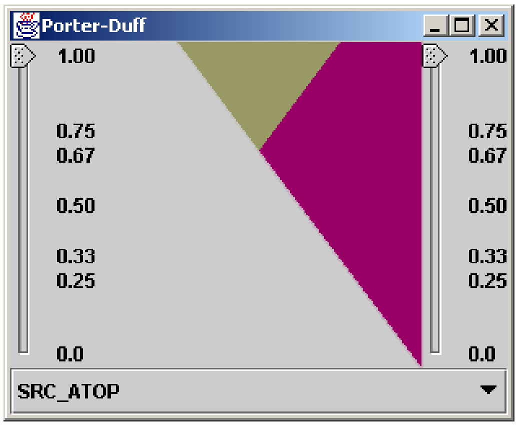
Rule 9.`AlphaComposite.DST_OUT`: Only the part of the destination image that doesn't overlap the source image will be drawn.

AlphaComposite.DST_OUT



Rule 10.`AlphaComposite.SRC_ATOP`*: The part of the source image that overlaps the destination image will be drawn with the part of the destination image that doesn't overlap the source image.

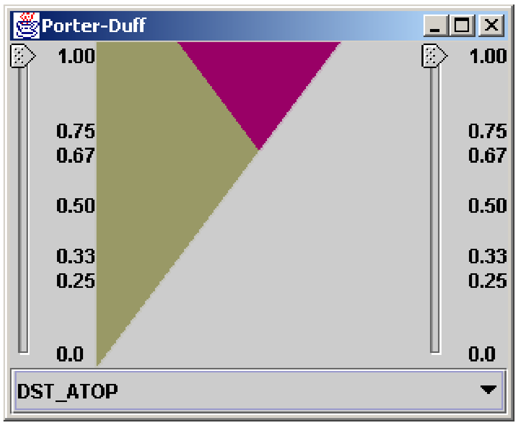
AlphaComposite.SRC_ATOP



SRC_ATOP

Rule 11.`AlphaComposite.DST_ATOP`*: The part of the destination image that overlaps the source image will be drawn with the part of the source image that doesn't overlap the destination image.

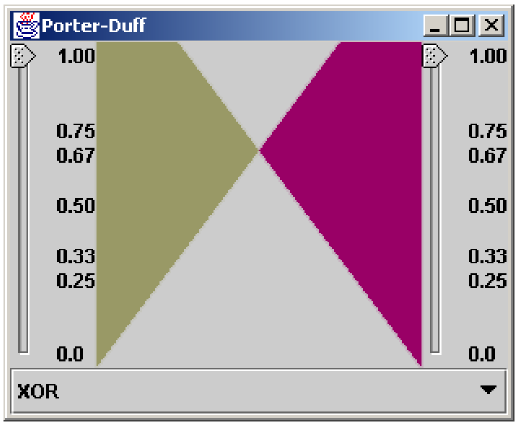
AlphaComposite.DST_ATOP



DST_ATOP

Rule 12.`AlphaComposite.XOR`*: The part of the source image that doesn't overlap the destination image will be drawn with the part of the destination image that doesn't overlap the source image.

AlphaComposite.XOR



XOR

A complete example

The following program is an interactive demonstration of the alpha compositing rules. Just alter the opacity for each triangle and select a rule to use to see the effect of blending the images.

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.lang.reflect.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;

public class CompositeIt extends JFrame {
  JSlider sourcePercentage = new JSlider();
  JSlider destinationPercentage = new JSlider();
  JComboBox alphaComposites = new JComboBox();
  DrawingPanel drawingPanel = new DrawingPanel();

  public CompositeIt() {
    super("Porter-Duff");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel contentPane = (JPanel) this.getContentPane();
    Dictionary labels = new Hashtable();
    labels.put(new Integer(0),   new JLabel("0.0"));
    labels.put(new Integer(25),  new JLabel("0.25"));
    labels.put(new Integer(33),  new JLabel("0.33"));
    labels.put(new Integer(50),  new JLabel("0.50"));
    labels.put(new Integer(67),  new JLabel("0.67"));
    labels.put(new Integer(75),  new JLabel("0.75"));
    labels.put(new Integer(100), new JLabel("1.00"));
    sourcePercentage.setOrientation(JSlider.VERTICAL);
    sourcePercentage.setLabelTable(labels);
    sourcePercentage.setPaintTicks(true);
    sourcePercentage.setPaintLabels(true);
    sourcePercentage.setValue(100);
    sourcePercentage.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        int sourceValue = sourcePercentage.getValue();
        drawingPanel.setSourcePercentage(sourceValue/100.0f);
      }
    });
    destinationPercentage.setOrientation(JSlider.VERTICAL);
    destinationPercentage.setLabelTable(labels);
    destinationPercentage.setPaintTicks(true);
    destinationPercentage.setPaintLabels(true);
    destinationPercentage.setValue(100);
    destinationPercentage.addChangeListener(new ChangeListener() {
      public void stateChanged(ChangeEvent e) {
        int destinationValue = destinationPercentage.getValue();
        drawingPanel.setDestinationPercentage(destinationValue/100.0f);
      }
    });
    String rules[] = {
      "CLEAR",    "DST",
      "DST_ATOP", "DST_IN",
      "DST_OUT",  "DST_OVER",
      "SRC",      "SRC_ATOP",
      "SRC_IN",   "SRC_OUT",
      "SRC_OVER", "XOR"};
    ComboBoxModel model = new DefaultComboBoxModel(rules);
    alphaComposites.setModel(model);
    alphaComposites.setSelectedItem("XOR");
    alphaComposites.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String alphaValue = alphaComposites.getSelectedItem().toString();
        Class alphaClass = AlphaComposite.class;
        try {
          Field field = alphaClass.getDeclaredField(alphaValue);
          int rule = ((Integer)field.get(AlphaComposite.Clear)).intValue();
          drawingPanel.setCompositeRule(rule);
        } catch (Exception exception) {
          System.err.println("Unable to find field");
        }
      }
    });
    contentPane.add(sourcePercentage, BorderLayout.WEST);
    contentPane.add(destinationPercentage, BorderLayout.EAST);
    contentPane.add(alphaComposites, BorderLayout.SOUTH);
    contentPane.add(drawingPanel, BorderLayout.CENTER);
    pack();
  }

  public static void main(String args[]) {
    new CompositeIt().show();
  }

  class DrawingPanel extends JPanel {
    GeneralPath sourcePath, destPath;
    BufferedImage source, dest;
    float sourcePercentage = 1, destinationPercentage = 1;
    int compositeRule = AlphaComposite.XOR;
    Dimension dimension = new Dimension(200, 200);

    public DrawingPanel() {
      sourcePath = new GeneralPath();
      sourcePath.moveTo(0,   0);   sourcePath.lineTo(150, 0);
      sourcePath.lineTo(0, 200);   sourcePath.closePath();
      source = new BufferedImage(200, 200, BufferedImage.TYPE_INT_ARGB);
      destPath = new GeneralPath();
      destPath.moveTo(200,  0);    destPath.lineTo(50, 0);
      destPath.lineTo(200, 200);    destPath.closePath();
      dest = new BufferedImage(200, 200, BufferedImage.TYPE_INT_ARGB);
    }

    public void setSourcePercentage(float value) {
      sourcePercentage = value;
       repaint();
    }

    public void setDestinationPercentage(float value) {
      destinationPercentage = value;
       repaint();
    }

    public void setCompositeRule(int value) {
      compositeRule = value;
       repaint();
    }

    public void paintComponent(Graphics g) {
      super.paintComponent(g);
      Graphics2D g2d = (Graphics2D)g;
      Graphics2D sourceG = source.createGraphics();
      Graphics2D destG = dest.createGraphics();

      destG.setComposite(AlphaComposite.Clear);
      destG.fillRect(0, 0, 200, 200);
      destG.setComposite(AlphaComposite.getInstance(
          AlphaComposite.XOR, destinationPercentage));
      destG.setPaint(Color.magenta);
      destG.fill(destPath);
      sourceG.setComposite(AlphaComposite.Clear);
      sourceG.fillRect(0, 0, 200, 200);
      sourceG.setComposite(AlphaComposite.getInstance(
        AlphaComposite.XOR, sourcePercentage));
      sourceG.setPaint(Color.green);
      sourceG.fill(sourcePath);

      destG.setComposite(AlphaComposite.getInstance(compositeRule));
      destG.drawImage(source, 0, 0, null);
      g2d.drawImage(dest, 0, 0, this);
    }

    public Dimension getPreferredSize() {
      return dimension;
    }
  }
}
```

Downloadable resources

Description	Name	Size
	j-mer0918.zip	2KB

Related topics

- Read about the different Porter-Duff rules in the original SigGraph paper.
- IBM does extensive research in the areas of graphics and visualization. Check out some of the projects currently in the works.
- Read the complete collection of Merlin tips by John Zukowski.