JOTSA Collections

Steven Robbins

November, 1996

Technical Report CS-97-xx

Abstract. This technical report describes how JOTSA handles grouping objects to that they can be manipulated together.

Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249

1 The problem

In animations it is sometimes convenient to consider a collection of objects as a group which can be manipulated as a unit. You can think of this as a method of making JOTSA objects which are more complicated than the basic object types.

2 How it works

In JOTSA when an object, called the slave, is linked to another object, called the master, certain operations applied to the master affect the slave. These include operations involving position and size. When the master is moved, the slave follows. When the master has its size changed with scaling factors, the slave is also scaled. JOTSA supports separate scaling factors in both the x- and y-directions.

When a slave is linked to a master, the slave's position is specified relative to the master, rather than in absolute terms. When the master is scaled, the relative position of each of its slaves is also scaled. Since the position of most JOTSA objects corresponds to the center of the object, scaling an object affects objects linked to it in a natural way.

3 Some examples

Figure 1 shows two views of stacked rectangles. The blue rectangle is on top of the red one. On the left they are shown normal size. Both rectangles have the same size and the distance between their centers is the same as their height.

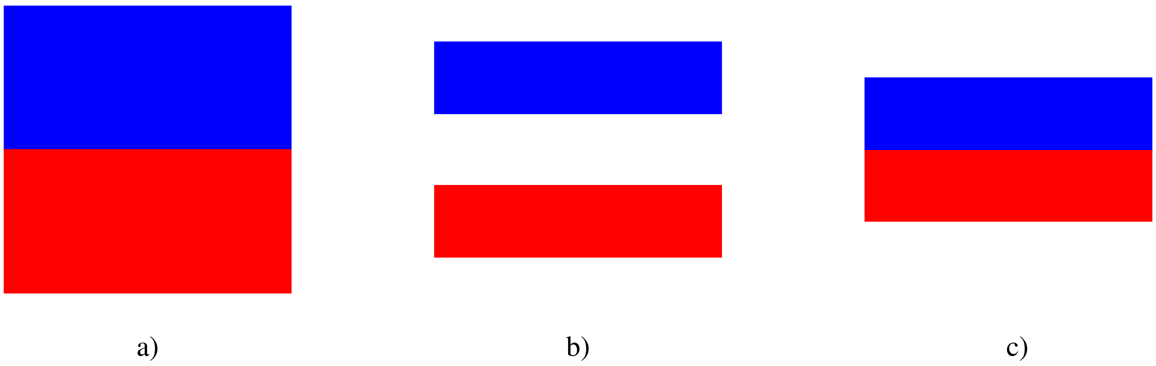
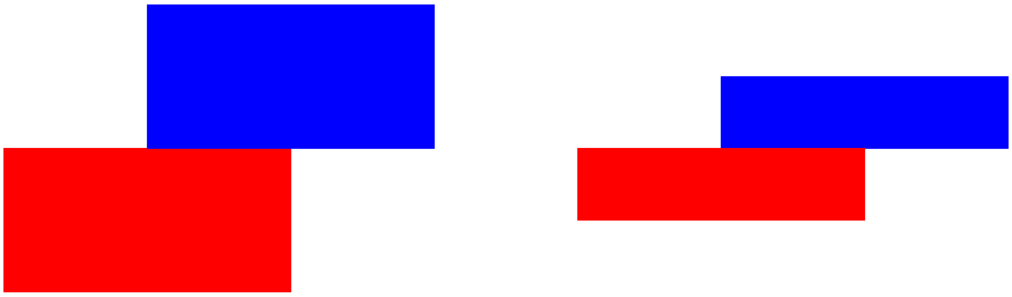


Figure 1: Two rectangles, one on top of the other. In diagram a) they are shown normal size. Diagram b) shows the result of scaling the two rectangles individually by a factor of 1/2 in the y-direction. Diagram c) shows the result of the same scaling on the collection.

Suppose the top rectangle (the slave) is linked to the bottom one, the master. The position of the top rectangle is specified by the distance between its center and the center of the master. If the master is scaled in the y-direction by a factor of 1/2, the slave will also be scaled and its distance to the master will be reduced. The result is shown on the right. Scaling the master has the effect of scaling the collection in a natural way.

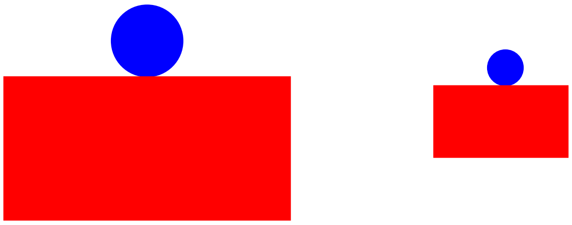
Figure 2 shows a similar situation with two rectangles with the center of one directly above the right edge of the other.



a) b)

Figure 2: Two rectangles, one on top of the other.

Figure 3 shows a similar situation with a rectangle and circle.

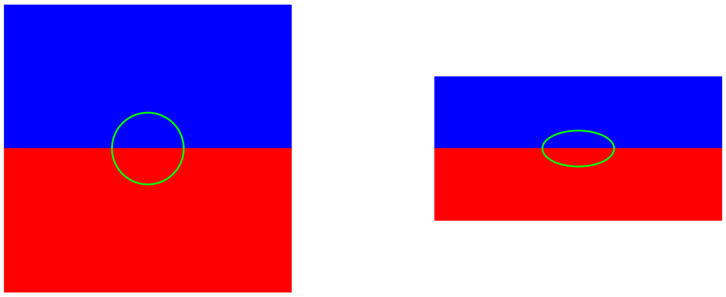


a) b)

Figure 3: A rectangle with a circle on top.

In each of these cases, one of the two objects was distinguished as the master and the other as the slave. In fact, their roles could have been reversed. A more symmetric treatment would have both objects as slaves linked to a master at the center of the collection. This is shown in in the left part of Figure 4. The master object is a small green circle. The two slave rectangles have their centers at the same position as the center of the master. On the right is shown the situation after the master has been scaled in the y-direction by a factor of 1/2. If we ignore the master object, it looks identical to diagram on the right side of Figure 1.

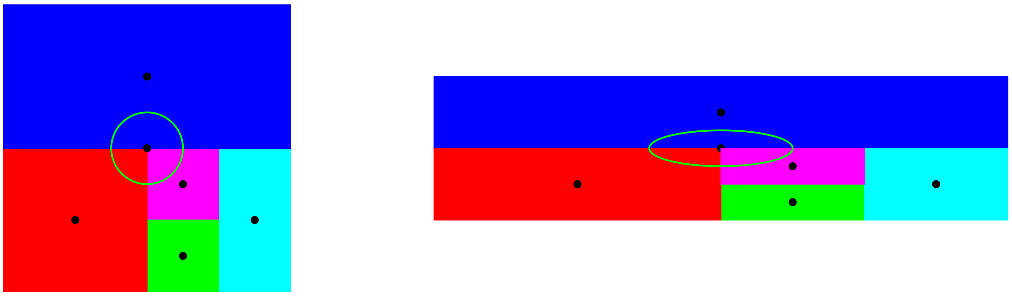
Figure 5 shows a more complicated configuration. The circle is the in the center of the collection of a number of rectangles of various sizes. On the right is shown then configuration scaled by a factor of 2 in the x-direction and by 1/2 in the y-direction. As long as



a) b)

Figure 4: Two rectangles, one on top of the other.

the centers of all of the rectangles are given relative to the circle, and the dimensions of the rectangles are simultaneously scaled when the center is scaled.



a) b)

Figure 5: A collection of rectangles.

The key to successful scaling is that the position of an object be specified in terms of the collection's center. For this to work, both the object and the collection must have a natural notion of center. This is true for rectangles, ovals and strings. Lines are handled differently in JOTSA, as the two endpoints are handled independently. It is the centers of the endpoints that need to be manipulated.

Show a figure with two rectangles connects with a line segment having an arrow.

4 Making Collections in JOTSA

JOTSA provides a class called `JotsaAnimationCollection` for specifying a collection of objects linked to a given master object. The collection is manipulated by manipulating the master object. The master can be moved or scaled and the collection will move or scale with it. The constructor is

```
public JotsaAnimationCollection(JotsaAnimationObject master);
```

The following methods in the `JotsaAnimationCollection` class are available:

```
public JotsaAnimationObject get_master();
```

which returns the master object of the collection.

```
public void insert(JotsaAnimationObject obj);
```

which inserts an object in the collection. This links the object to the master of the collection. The object will have the same position as the master.

```
public void insert(JotsaAnimationObject obj, int xoff, int yoff);
```

which inserts an object in the collection while setting the relative position of the object form the master.

```
public void set_scale(double s);
```

which scales the collection by scaling the master. This scales in both the x-direction and y-direction.

```
public void set_scale(double sx, double sy);
```

which scales the collection by scaling the master. The x-direction scaling and the y-direction scaling can be separately set.

```
public void scale_scale(double s);
```

which scales the collection by scaling the master. The new scaling is applied to the old one so that the new scale factor is the old one multiplied by `s`.

```
public void scale_scale(double sx, double sy);
```

which scales the collection by scaling the master. This is similar to the one above except that the scale factors for the two dimensions can be set separately.

```
public JotsaAnimationCollection duplicate();
```

which makes a new collection which is identical to the old one. The master and all of the objects in the collection are duplicated so that changing and object in the new collection has no effect on the old collection.

```
public void merge(JotsaAnimationCollection col);
```

which combines two collections. All of the objects in the collection `col` except the master are moved into the current collection. The scalings and positions of the objects in `col` are modified so that they do not change their absolute size or position. If this is called while the master of `col` is moving, the final position of `col` is used.

5 An example illustrating merges and splits

5.1 The merge-split problem

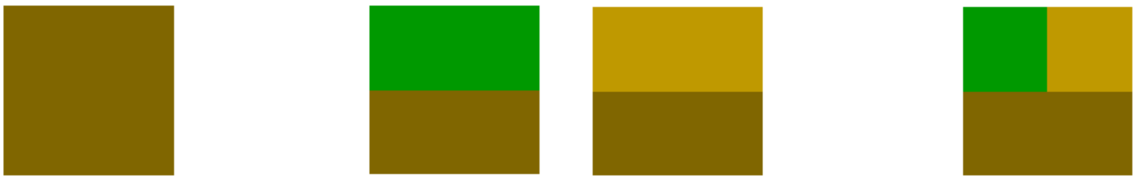
5.2 A merge-split visualization

In this visualization a cell is represented by a square. Initially the square has a single color. When it is split, it is shrunk vertically into a rectangle of half the original height and duplicated. Two new rectangles of different colors are created and each is put on one of the copies of the duplicated rectangle, forming two squares. Each one has a solid rectangle on top.

To merge two squares, each is shrunk horizontally by a factor of 2 and the two resulting rectangles are put side by side forming a single square.

Figure 6a) shows an initial cell before any splitting or merging. Figure 6a) show the result have this cell splits. Figure 6c) Show the result of the two cells in b) merging. It is possible to recover the entire split-merge history from this cell representation. If the top half of a cell is a rectangle of a single color, the last operation performed on the cell was a split. The entire history of the cell before the split occurred is contained in the lower half of the cell. If the top half is not a solid rectangle, the cell was generated by merging two cells and the left and right half each contain a complete history of the one of the two cells that merged.

For example, the left side of Figure 5 represents a cell which just underwent a split since the top half of the square is a rectangle of a single color. The bottom half represents the history of the cell before it was split. Since the top half of this (the top half of the bottom half) is not of a solid color, that cell was formed by the merging of two cells, one of which (the left part) had no previous split or merge operations. The other cell is represent by the lower left quarter of the square and was the result of merging two cells, one of which had been split and the other had no previous splits or merges. while the other



a) b) c)

Figure 6: Diagram a) shows an initial cell before any splitting or merging. Diagram b) shows the result after a split. Diagram c) shows the result of merging the two cells from b).

5.3 An applet to visualize merges and splits

This section describes an applet which shows how JOTSA can be used to visualize splits and merges. Figure 7 shows the applet display after a single split has occurred.

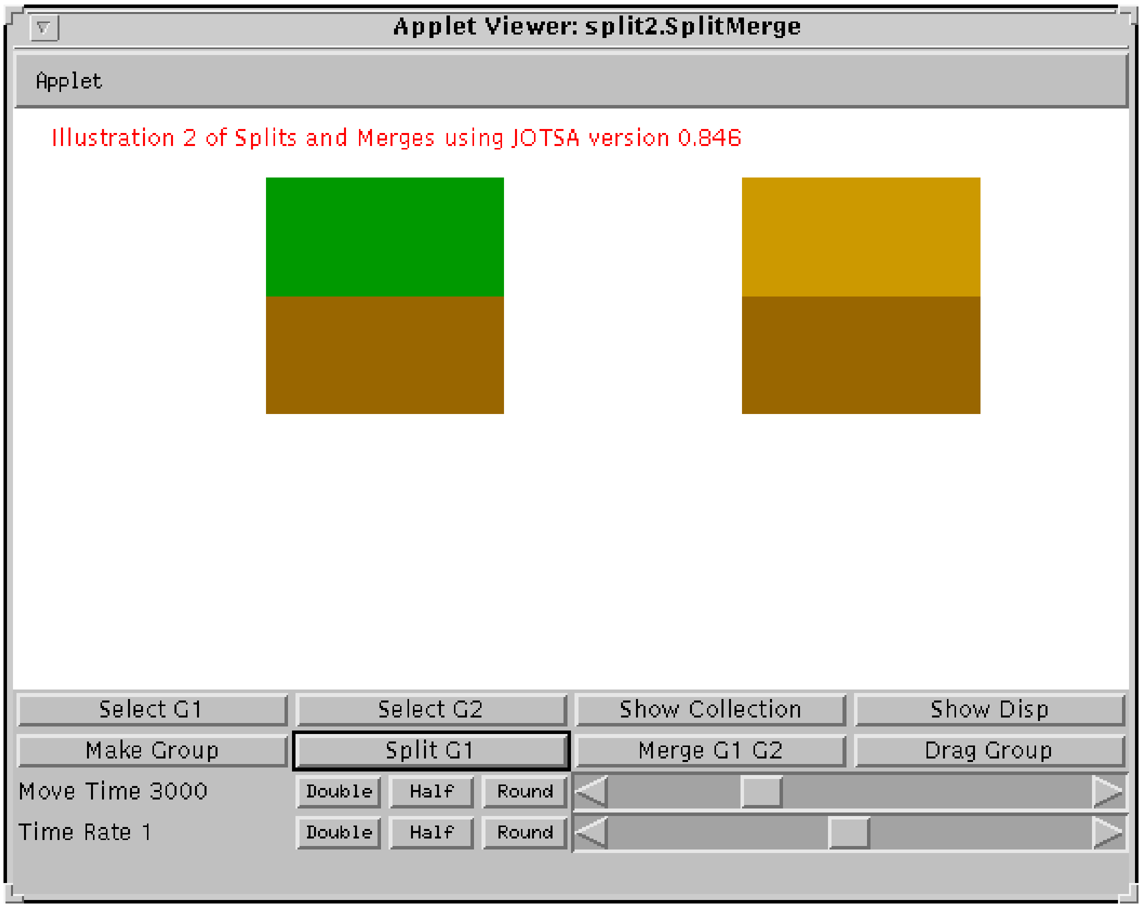


Figure 7: The applet to illustrate splits and merges. It is shown after a single split.

To create a new cell, click on *Make Group*. A square with a solid color appears. To split a cell, first select the cell as group 1, and click on *Split G1*. A cell is selected as group 1 by clicking *Select G1* and then clicking near the cell. The cell whose center is closest to the place where the mouse was clicked is selected as group 1. The selected group 1 cell is shown with a small green circle at its center.

After the split there are two new cells side by side. To split one of these again, select it as group 1 and click *Split G1* again.

The cells can be dragged around with the mouse. To do this, click *Drag Group* and click the mouse button near the cell you wish to move. The cell whose center is closest to the mouse when it was clicked will follow the mouse until the button is released.

To merge two cells, select one as group 1 by clicking on *Select G1* and clicking near the cell to be selected. Now select a cell to be group 2 by clicking on *Select G1* and clicking near that cell. The cell selected as group 2 is shown with a blue circle at its center. Merge the cells by clicking on *Merge G1 G2*.

The other two buttons are mainly for debugging. The `Show Collection` button displays a list of the current collections and the `Show Disp` button displays the number of JOTSA objects currently displayed. Output for each goes to the Java console.

The two sliders on the bottom of the screen represent the time it takes to do a split or merge in milliseconds, and the rate at which virtual time moves.

5.4 Implementation Details

In this application, a cell is represents by a collection of JOTSA objects. Each collection has a master object at its center which is a circle. Normally this master is not displayed, but is shown when the object is selected as group 1 or group 2. The `JotsaAnimationObject` methods `set_inhibit_display` and `clear_inhibit_display` are used to make this master object visible or invisible. The color is changed by `set_color`. A cell is represented by the master and a collection of squares all of the same size. These squares are scaled into rectangles and their positions relative to the center of the master are set so they form a collection of rectangles that are fitted snugly together.

The class `SplitMergeObjects` manages the collections. It keeps a list of collections and has methods for inserting and removing collections from the list. There are also methods for determining the position of a collection and which collection is closest to a given point. It also contains methods for creating master objects and the pieces that make up the collections. It uses the JOTSA `ColorList` utility to choose colors for the various parts of the collections. It is described fully in Section 6.

The *Drag Group* operation is accomplished just by moving the master. *Make Group* makes a new collection containing a master circle and a solid rectangle. The groups are kept in a vector. The vector is scanned when it needs to be determined which of the collections is closest to the mouse.

5.4.1 The Split

The code to implement *Split G1* is shown in Figure 8. The split thread is passed the applet, the set of objects, the collection to move and the time to move in each step. A *Split G1* is accomplished in 5 steps.

Step 1. Shrink the collection by a factor of 2 in the y-direction and wait until it is done. The center of the master must move along down as the scale is changed in such a way the the bottom edge of the collection does not move. The movement is done with `path_create_along_line` with the x-coordinate unchanged and the y-coordinate moving down by one fourth of the size of the object. As it moves the scale in the y-direction is changed for 1.0 to 0.5 using `set_scale_linear`. The time to move is set using `times_set`. `JotsaForceRedisplay` causes JOTSA to redisplay immediately. The thread then waits for the movement to complete using `JotsaWait()`. This atomicly activates the object, instructs the object to notify the thread when it is done moving, and suspends the thread. When the thread wakes up the master has a scaling factor of 1/2 in the y-direction and has moved down a bit.

Step 2. Move everything into a new collection which has a scale factor of 1.0. The master will be always kept so that it has a scale factor of 1 in both dimensions when it is not undergoing a split or merge. We could accomplish this by adding additional scaling to each object in the collection and the resetting the scale of the master. Unfortunately, this causes the pieces to temporarily have the wrong size. So instead, we create a new collection with an unscaled master and move each object from the old collection to the new one. As an object is moved, its scale and position relative to the master are adjusted so that it stays the same place with the same size. All of this is accomplished with the `merge` method of the `JotsaAnimationCollection` class.

Step 2 starts by removing the old collection from the list of collections. Then the method `reset_to_final` is used to set the master's position to its final position along the path that it has moved. A new master and a new collection are then created. The `merge` method then moves all of the objects from the old collection into this new one. The new collections is added to the list of collections and the old master is removed from the list of displayed objects.

Step 3. Make a collection which is identical to this one. This is the split operation. The collection is duplicated using the `duplicate` method of the `JotsaAnimationCollection` class. This is accomplished by making a new master and a new collection, making a copy of each object in the collection and linking it to the new master in the same way as the original. This is done with the `duplicate` method of the `JotsaAnimationObject` class. When a `JotsaAnimationObject` is duplicated, copies of all reference variables are made so that the two objects can be independently modified.

Step 4. Put different colored rectangles on top of each collection. A new rectangle is inserted in each collection to make a full square. The method `create_split_object` makes a square. This object is then scaled in the y-direction by a factor of 1/2. This procedure is done once for each of the two collections.

Step 5. Move the new collection to the left a distance twice the width of the collection and wait for completion. The movement is done in a way similar to Step 2 and `JotsaWait` is again used to wait until the motion is complete. The position of the new master is then set to its final position.

In summary, all of the objects in the collections (except the master) are squares with the appropriate scaling factors. When the collection is shrunk, it is done by shrinking the master. After the shrinking is complete, a new collection is created with unit scaling factors and the objects from the old collection are merged into the new one.

5.4.2 The Merge

The code to implement the *Merge G1 G2* thread is shown in Figure 9. The Merge thread is passed the applet, the set of all objects, the two collections to merge, and the speed at which to move the objects during the merge. `rsize` is the size in pixels of one of the collections.

Step 1. Shrink each collection horizontally while moving them together. First the initial

```
package split2;

import java.awt.*;
import java.awt.image.*;
import java.util.Vector;
import jotsa.*;
import jotsa.utility.*;

public class Split extends JotsaWaitingThread {

   JotsaAnimationCollection col;
   JotsaAnimationApplet ap;
   SplitMergeObjects objs;
   int speed_value;

   public Split(JotsaAnimationApplet ap, SplitMergeObjects objs,
                JotsaAnimationCollection col, int speed_value) {
      super(ap);
      this.ap = ap;
      this.objs = objs;
      this.col = col;
      this.speed_value = speed_value;
      start();
   }

   public void run() {
      int init_x;
      int init_y;
      int rsize;
      JotsaAnimationObject master;
      JotsaAnimationObject mastertemp;
      JotsaAnimationObject tempanim;
      JotsaAnimationCollection coltemp;
      JotsaAnimationCollection colnew;
      JotsaAnimationObject masternew;

      if (col == null) return;
      rsize = objs.get_rect_size();
      master = col.get_master();
      init_x = master.get_firstx();
      init_y = master.get_firsty();

/* Step 1: Shrink the collection by a factor of 2 and wait until done. */
      master.path_create_along_line(init_x,init_y,init_x,init_y+rsize/4);
      master.set_scale_linear(1.0,1.0,1.0,0.5);
      master.times_set(speed_value);
      ap.JotsaForceRedisplay();
      JotsaWait(master);

/* Step 2: Move everything into a new collection which is not scaled. */
      objs.remove_collection(col);
      master.reset_to_final();
      mastertemp =
         objs.create_master_object(init_x,init_y,Color.blue);
      coltemp = new JotsaAnimationCollection(mastertemp);
      col.merge(coltemp);
      objs.add_collection(coltemp);
      ap.JotsaRemoveObject(master);

/* Step 3: Make another collection which is identical to this one. */
      colnew = coltemp.duplicate();
      objs.add_collection(colnew);


/* Step 4: Put different colored rectangles on top of each collection. */
      tempanim = objs.create_split_object();
      tempanim.set_scale(1.0,0.5);
      coltemp.insert(tempanim,0,-rsize/2);
      tempanim = objs.create_split_object();
      tempanim.set_scale(1.0,0.5);
      colnew.insert(tempanim,0,-rsize/2);

/* Step 5: Move the new collection to the left. */
      masternew = colnew.get_master();
      masternew.path_create_along_line(init_x,init_y,init_x+2*rsize,init_y);
      masternew.times_set(speed_value);
      ap.JotsaForceRedisplay();
      JotsaWait(masternew);
      masternew.reset_to_final();
   }

}
```

Figure 8: A thread to implement a split.

positions of the two collections are determined. The final position of the merged object is half way between the two. When each master is scaled to have half its original width, the two collections will be adjacent if their centers are `rsize/2` apart. A path for each master is set so it will end up adjacent to the other, half bay between their original positions. They are each set to scale as they move. The time to move the first master is set to `speed_value` and the second master is set to move along with it with `path_set_index_linked`. The screen is redisplayed and `JotsaWait` is used to activate the first master and wait for it to complete its motion. Since the second master is index linked to the first, it starts moving when the first one does. When the motion is complete, each master is set to have its position be its final position and the second master is unlinked from the first.

Step 2. Merge the two collections into a new one with unit scaling. A new master and collection are created. Each of the old collections is merged into the new one with `merge` and the old collections and masters are removed.

```
package split2;

import java.awt.*;
import java.awt.image.*;
import java.util.Vector;
import jotsa.*;

public class Merge extends JotsaWaitingThread {

   JotsaAnimationCollection col1;
   JotsaAnimationCollection col2;
   SplitMergeObjects objs;
   JotsaAnimationApplet ap;
   int speed_value;

   public Merge(JotsaAnimationApplet ap, SplitMergeObjects objs,
                                JotsaAnimationCollection col1,
                                JotsaAnimationCollection col2,
                                int speed_value) {
      super(ap);
      this.ap = ap;
      this.objs = objs;
      this.col1 = col1;
      this.col2 = col2;
      this.speed_value = speed_value;
      start();
   }

   public void run() {
      int init1_x;
      int init1_y;
      int init2_x;
      int init2_y;
      int final_x;
      int final_y;
      int rsize;
      JotsaAnimationObject master1;
      JotsaAnimationObject master2;
      JotsaAnimationObject newmaster;
      JotsaAnimationCollection newcol;

      if (col1 == null) return;
      if (col2 == null) return;
      rsize = objs.get_rect_size();
      master1 = col1.get_master();
      master2 = col2.get_master();
      master1.reset_to_final();
      master2.reset_to_final();

/* Step 1: Shrink each collection horizontally while moving them together. */
      init1_x = master1.get_firstx();
      init1_y = master1.get_firsty();
      init2_x = master2.get_firstx();
      init2_y = master2.get_firsty();
      final_x = (init1_x + init2_x)/2;
      final_y = (init1_y + init2_y)/2;
      master1.path_create_along_line(init1_x,init1_y,
                                    final_x-rsize/4,final_y);
      master2.path_create_along_line(init2_x,init2_y,
                                    final_x+rsize/4,final_y);
      master1.set_scale_linear(1.0, 0.5, 1.0, 1.0);
      master2.set_scale_linear(1.0, 0.5, 1.0, 1.0);
      master1.times_set(speed_value);
      master2.path_set_index_linked(master1);
      ap.JotsaForceRedisplay();
      JotsaWait(master1);
      master1.reset_to_final();
      master2.reset_to_final();
      master2.path_clear_index_linked();

/* Step 2: Merge the two collections into a new one with unit scaling. */
      newmaster =
         objs.create_master_object(final_x,final_y,Color.blue);
      newcol = new JotsaAnimationCollection(newmaster);
      col1.merge(newcol);
      objs.remove_collection(col1);
      ap.JotsaRemoveObject(master1);
      objs.add_collection(newcol);
      col2.merge(newcol);
      objs.remove_collection(col2);
      ap.JotsaRemoveObject(master2);
      ap.JotsaForceRedisplay();
   }

}
```

Figure 9: A thread to implement a merge.

6 Managing the Collections

The `SplitMergeObjects` class is used to manage the collections and objects that comprise them. It keeps a list of all of the collections and provides methods for creating new masters, collections, and objects.

The constructor just takes the applet as a parameter. It sets up an empty list of collections and initializes a JOTSA `ColorList` object to the default. This allows for 125 distinct colors. The `ColorList` is used to obtain unique colors when a new piece is added to a collection. It is described in detail in Section 7.

The `add_collection` and `remove_collection` methods add and remove collections from the internal list. The `get_rect_size` method returns the size of a collection. The is the number of pixels in its width or height.

The method `get_closest_collection` returns the collection from the list that is closest to the given point and `get_closest_collection_position` returns a `Point` which is at the center of the closest collection.

When a split is done, each collection needs to have a rectangle of a unique color added to it. The method `create_split_object` creates and returns a `JotsaAnimationObject` which is a square of the correct size. `create_master_object` creates and returns an oval which has its display inhibited. `create_solid_object` creates and returns a filled oval which is used for the `Select` and `Drag` operations.

The method `hide_master` causes the master to not be displayed. It will be displayed when its collection is selected. This is done with `show_maser`. To make sure the master is displayed on top of the objects in its collections, rather than hidden behind them, its level is set to be greater than anything yet created. The level of an object cannot be changed while it is in JOTSA's list of displayed objects, so first it is removed from that list, the level is changed, and then it is reinserted.

The code for `SplitMergeObjects` is shown below.

```
package split2;

import java.awt.*;
import java.awt.image.*;
import java.util.Vector;
import jotsa.*;
import jotsa.utility.ColorList;

public class SplitMergeObjects {
   Vector clist;
   private int rectsize = 128;
   private int master_disp_size = 30;
   private ColorList Colors;
   JotsaAnimationApplet ap;

   public SplitMergeObjects(JotsaAnimationApplet ap) {
      this.ap = ap;
      Colors = new ColorList();
      Colors.CurrentColorSet(Color.red);
      clist = new Vector();
   }

   public void add_collection(JotsaAnimationCollection col) {
      clist.addElement(col);
   }

   public void remove_collection(JotsaAnimationCollection col) {
      clist.removeElement(col);
   }


   public int get_rect_size() {
      return rectsize;
   }
```

```
public JotsaAnimationCollection get_closest_collection(Point p) {
   int shortest;
   int thisdist;
   int size;
   Point pos;
   JotsaAnimationCollection closest;
   JotsaAnimationCollection col;
   size = clist.size();
   if (size == 0) return null;
   closest = get_from_clist(0);
   if (size == 1) return closest;
   pos = get_collection_position(closest);
   shortest = dsquare(p,pos);
   for (int i=1;i<size;i++) {
      col = get_from_clist(i);
      pos = get_collection_position(col);
      thisdist = dsquare(p,pos);
      if (thisdist < shortest) {
         closest = col;
         shortest = thisdist;
      }
   }
   return closest;
}

public Point get_closest_collection_position(Point p) {
   JotsaAnimationCollection col;
   col = get_closest_collection(p);
   if (col == null) return null;
   return get_collection_position(col);
}

public static Point get_collection_position(JotsaAnimationCollection col) {
   JotsaAnimationObject master;
   master = col.get_master();
   return new Point(master.get_firstx(),master.get_firsty());
}

public JotsaAnimationObject create_split_object() {
   JotsaAnimationObject ani;
   int current_level;
   current_level = ap.JotsaNextLevel();
   ani = new JotsaAnimationObject(0,0,current_level,current_level,ap);
   ani.set_fill_centered_rect(rectsize,rectsize,Colors.NextColor());
   ap.JotsaInsertObject(ani);
   return ani;
}

public JotsaAnimationObject create_master_object(int x, int y,
                                                    Color C) {
   JotsaAnimationObject ani;
   int current_level;
   current_level = ap.JotsaNextLevel();
   ani = new JotsaAnimationObject(x,y,current_level,current_level,ap);
   ani.set_draw_centered_oval(master_disp_size,master_disp_size,C);
   ani.set_inhibit_display();
   ap.JotsaInsertObject(ani);
   return ani;
}

public JotsaAnimationObject create_solid_object(int x, int y,
                                                   int level, Color C) {
   JotsaAnimationObject ani;
   ani = new JotsaAnimationObject(x,y,level,level,ap);
   ani.set_fill_centered_oval(master_disp_size,master_disp_size,C);
   ani.set_inhibit_display();
   ap.JotsaInsertObject(ani);
   return ani;
}

public void show_master(JotsaAnimationCollection col) {
   JotsaAnimationObject master;
   master = col.get_master();
   ap.JotsaRemoveObject(master);
   master.set_level(ap.JotsaNextLevel());
   ap.JotsaInsertObject(master);
   master.clear_inhibit_display();
}

public void hide_master(JotsaAnimationCollection col) {
   JotsaAnimationObject master;
   master = col.get_master();
   if (master == null) return;
   master.set_inhibit_display();
}

public void show_clist() {
   int size;
   size = clist.size();
   System.out.println("Collection size: "+size);
   for (int i=0;i<size;i++)
      System.out.println("   "+i+": "+get_from_clist(i));
```

```
   }

   private JotsaAnimationCollection get_from_clist(int i) {
      return (JotsaAnimationCollection)clist.elementAt(i);
   }


   private int dsquare (Point p1, Point p2) {
      return (p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y)*(p1.y-p2.y);
   }

}
```

7 The JOTSA ColorList

The `ColorList` is in the `jotsa.utility` package. It provide a method for obtaining unique colors from a simply-defined set. You specify a set of color values in the range 0 to 155, and the `ColorList` will generate all colors that use the set of values. By default, the following five values are used if the constructor is called with no parameters: 0, 102, 153, 204, 255. Alternatively, the constructor can be given an array of color values to use.

The colors are treated as if they are in a three-dimensional array in row-major form with red in the first dimension, green in the second, and blue in the third. Since adjacent colors may look similar, when colors are created, a stride is used. If the stride does not have any. If the stride is odd, all colors are generated. The default stride is 3, but it may be changed with `ColorIncrementSet`. The value can be obtained with `ColorIncrementGet`.

At any time there is a current color which can be returned with `CurrentColor` and set with `CurrentColorSet`. The current color can be changed to the next color and this color returned with `NextColor`.

A random color from the list can be generated and returned with `RamdomColorGet`. `RandomColorSet` is similar but also sets the current color to the random color which is returned.

The array of color values being used can be returned with `ColorListGet` and the default list can be gotten with `DefaltColorListGet`.

```
package jotsa.utility;

import java.awt.*;
import java.awt.image.*;
import java.util.*;

public class ColorList {

   private static int[] default_colorlist={0, 102, 153, 204, 255};
   private int[] colorlist;
   private int colorlistsize;
   private int next_color_increment;
   private Color current_color;

   public ColorList(int[] clist) {
      int firstind;
      colorlistsize = clist.length;
      colorlist = new int[colorlistsize];
      for (int i=0;i<colorlistsize;i++)
         colorlist[i] = clist[i];
      next_color_increment = 3;
      firstind = clist[0];
      current_color = new Color(firstind,firstind,firstind);
   }

   public ColorList() {
      int firstind;
      colorlistsize = default_colorlist.length;
      colorlist = new int[colorlistsize];
      for (int i=0;i<colorlistsize;i++)
         colorlist[i] = default_colorlist[i];
      next_color_increment = 3;
      firstind = default_colorlist[0];
      current_color = new Color(firstind,firstind,firstind);
   }

   public int ColorIncrementGet() {
      return next_color_increment;
   }

   public void ColorIncrementSet(int inc) {
      next_color_increment = inc;
   }

   public int[] ColorListGet() {
      int[] new_colorlist;
      new_colorlist = new int[colorlistsize];
      for (int i=0;i<colorlistsize;i++)
         new_colorlist[i] = colorlist[i];
      return new_colorlist;
```

```
   }

/*    Private methods   */
   public Color CurrentColor() {
      return current_color;
   }

   public Color CurrentColorGet() {
      return current_color;
   }

   public void CurrentColorSet(Color C) {
      current_color = C;
   }

   public static int[] DefaultColorListGet() {
      int[] new_colorlist;
      new_colorlist = new int[default_colorlist.length];
      for (int i=0;i<default_colorlist.length;i++)
         new_colorlist[i] = default_colorlist[i];
      return new_colorlist;
   }

   public Color NextColor() {
      for (int i=0;i<next_color_increment;i++)
         current_color = GetNextColor(current_color);
      return current_color;
   }

   public Color RandomColorGet() {
      int cred,cgreen,cblue;

      randomize();
      cred = colorlist[(int)(colorlistsize*Math.random())];
      randomize();
      cgreen = colorlist[(int)(colorlistsize*Math.random())];
      randomize();
      cblue = colorlist[(int)(colorlistsize*Math.random())];
      return new Color(cred,cgreen,cblue);
   }

   public Color RandomColorSet() {
      int cred,cgreen,cblue;

      randomize();
      cred = colorlist[(int)(colorlistsize*Math.random())];
      randomize();
      cgreen = colorlist[(int)(colorlistsize*Math.random())];
      randomize();
      cblue = colorlist[(int)(colorlistsize*Math.random())];
      current_color = new Color(cred,cgreen,cblue);
      return current_color;
   }

/*    Private methods   */

   private void randomize() {
      for (int i=0;i<10;i++)
         Math.random();
   }

   private int find_color_index(int val) {
      for (int i=0;i<colorlistsize;i++)
         if (colorlist[i]==val) return i;
      return -1;
   }

   private Color GetNextColor(Color C) {
      int red_index;
      int green_index;
      int blue_index;
      red_index = find_color_index(C.getRed());
      green_index = find_color_index(C.getGreen());
      blue_index = find_color_index(C.getBlue());
      if ( red_index < 0) return new Color(0,0,0);
      if ( green_index < 0) return new Color(0,0,0);
      if ( blue_index < 0) return new Color(0,0,0);
      red_index++;
      if (red_index >= colorlist.length) {
         red_index = 0;
         green_index++;
      }
      if (green_index >= colorlist.length) {
         green_index = 0;
         blue_index++;
      }
      if (blue_index >= colorlist.length) {
         blue_index = 0;
      }
      return new Color(colorlist[red_index],colorlist[green_index],
                       colorlist[blue_index]);
   }
```

```
}
```

8 The Main Appet

Most of the code in the main applet handles the layout, the buttons and the dragging collections. All of the actual work in splitting and merging takes place in the two threads —mytt Split and `Merge`.

```
/*
   <Applet code = "split2.SplitMerge"
          width = 600 height = 400>
   </applet>
 */

package split2;

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.util.*;
import jotsa.*;

public class SplitMerge extends JotsaAnimationApplet {

   Image timage;
   int mouse_x;
   int mouse_y;
   int width;
   int height;
   SplitMergeObjects Collections;
   JotsaAnimationObject obj;
   JotsaSlider speed_control;
   JotsaSliderf time_control;
   String backstr1;
   boolean special_flag = false;
   JotsaAnimationCollection group;
   JotsaAnimationCollection group1;
   JotsaAnimationCollection group2;
   JotsaAnimationCollection select1;
   JotsaAnimationCollection select2;
   JotsaAnimationCollection drag_group;
   boolean select_group_1_flag;
   boolean select_group_2_flag;
   boolean drag_group_flag;
   int drag_offset_x;
   int drag_offset_y;
   int run_type;
   final int SELECTOR_LEVEL = 100000;
   JotsaAnimationObject timestring;
   JotsaAnimationObject selector_1;
   JotsaAnimationObject selector_2;
   JotsaAnimationObject selector_d;


   public void init() {
       super.init();
       width = bounds().width;
       height = bounds().height-120;
       setup_layout();
       JotsaInitImages();
       backstr1=
         "Illustration 2 of Splits and Merges using JOTSA version "+
          JotsaVersionMajor+"."+JotsaVersionMinor;
       JotsaWriteBackgroundString(backstr1,20,20,new Color(255,0,0));
       timestring = new JotsaAnimationObject(width-150,20,
                                   current_level,current_level,this);
       Collections = new SplitMergeObjects(this);
       selector_1 =
          Collections.create_solid_object(0,0,SELECTOR_LEVEL,Color.green);
       selector_2 =
          Collections.create_solid_object(0,0,SELECTOR_LEVEL,Color.blue);
       selector_d =
          Collections.create_solid_object(0,0,SELECTOR_LEVEL,Color.red);
       timestring.set_draw_string("",Color.black);
       JotsaInsertObject(timestring);
       select_group_1_flag = false;
       select_group_2_flag = false;
       drag_group_flag = false;
       drag_group = null;
       run_type = -1;
       JotsaForceRedisplay();
   }

   void setup_layout() {
       setLayout(new BorderLayout());
       Panel p = new Panel();
       Panel r = new Panel();
       Panel u = new Panel();
       p.setLayout(new GridLayout(4,1));
       r.setLayout(new GridLayout(1,4));
```

```
    u.setLayout(new GridLayout(1,4));
    speed_control = new JotsaSlider(3000,0,10000,1,200,"Move Time ",this);
    time_control = new JotsaSliderf(1000,0,2000,1000,200,"Time Rate ",this);
    r.add(new Button("Select G1"));
    r.add(new Button("Select G2"));
    r.add(new Button("Show Collection"));
    r.add(new Button("Show Disp"));
    u.add(new Button("Make Group"));
    u.add(new Button("Split G1"));
    u.add(new Button("Merge G1 G2"));
    u.add(new Button("Drag Group"));
    p.add(r);
    p.add(u);
    p.add(speed_control);
    p.add(time_control);
    add("South",p);
    add("Center",JotsaDefaultCanvas);
    validate();
}

String coordstr(int x, int y) {
   return "("+x+","+y+")";
}

void show_disp() {
   System.out.println("Number of displayed objects is "+JotsaNumObjects());
}

void clear_select_1() {
   if (select1 == null) return;
   select1.get_master().set_inhibit_display();
   select1 = null;
}

void clear_select_2() {
   if (select2 == null) return;
   select2.get_master().set_inhibit_display();
   select2 = null;
}

void set_select_1(Point p) {
   JotsaAnimationCollection col;
   col = Collections.get_closest_collection(p);
   if (col != null) set_select_1(col);
   selector_1.set_inhibit_display();
}

void set_select_2(Point p) {
   JotsaAnimationCollection col;
   col = Collections.get_closest_collection(p);
   if (col != null) set_select_2(col);
   selector_2.set_inhibit_display();
}

void set_select_1(JotsaAnimationCollection col) {
   if (select1 != null)
      Collections.hide_master(select1);
      select1 = col;
      col.get_master().reset_color(Color.green);
      Collections.show_master(col);
}

void set_select_2(JotsaAnimationCollection col) {
   if (select2 != null)
      Collections.hide_master(select2);
      select2 = col;
      col.get_master().reset_color(Color.blue);
      Collections.show_master(col);
}

Point get_collection_position(JotsaAnimationCollection col) {
   JotsaAnimationObject master;
   master = col.get_master();
   return new Point(master.get_firstx(),master.get_firsty());
}

void move_group_to_position(JotsaAnimationCollection col, int x, int y) {
   JotsaAnimationObject master;
   master = col.get_master();
   master.set_position(x,y);
}

public void JotsaChangeParameters() {
   JotsaSetRate(time_control.value()/1000.0);
   JotsaForceRedisplay();
}

public void JotsaPaintLocal(long tm) {
   timestring.reset_string(""+tm/1000.0);
}

public boolean JotsaHandleCanvasEvent(Event e) {
```

```
      int move_time;
      JotsaAnimationObject master;
      Point tp;
      int new_x, new_y;
   if (e.id == Event.MOUSE_DOWN) {
      showStatus("Down at "+coordstr(e.x,e.y));
      if (select_group_1_flag)
         set_select_1(new Point(e.x,e.y));
      else if (select_group_2_flag)
         set_select_2(new Point(e.x,e.y));
      else if (drag_group_flag) {
         drag_group = Collections.get_closest_collection(new Point(e.x,e.y));
         master = drag_group.get_master();
         drag_offset_x = e.x - master.get_firstx();
         drag_offset_y = e.y - master.get_firsty();
      }

      select_group_1_flag = false;
      select_group_2_flag = false;
      drag_group_flag = false;
   }
   else if (e.id == Event.MOUSE_UP) {
      showStatus("");
      mouse_x = e.x;
      mouse_y = e.y;
      selector_1.set_inhibit_display();
      selector_2.set_inhibit_display();
      selector_d.set_inhibit_display();
      if (drag_group == null) return true;
      move_group_to_position(drag_group,e.x-drag_offset_x,e.y-drag_offset_y);          JotsaForceRedisplay();
      drag_group = null;
   }
   else if (e.id == Event.MOUSE_MOVE) {
      mouse_x = e.x;
      mouse_y = e.y;
      tp = new Point(mouse_x,mouse_y);
      tp = Collections.get_closest_collection_position(tp);
      if (tp == null) {
         tp = new Point(mouse_x,mouse_y);
      }
      if (select_group_1_flag) {
         selector_1.set_position(tp.x,tp.y);
         JotsaForceRedisplay();
      }
      if (select_group_2_flag) {
         selector_2.set_position(tp.x,tp.y);
         JotsaForceRedisplay();
      }
      if (drag_group_flag) {
         selector_d.set_position(tp.x,tp.y);
         JotsaForceRedisplay();
      }
   }
   else if (e.id == Event.MOUSE_DRAG) {
      if (drag_group == null) return true;
      new_x = e.x-drag_offset_x;
      new_y = e.y-drag_offset_y;
      move_group_to_position(drag_group,new_x,new_y);
      selector_d.set_position(new_x,new_y);
      JotsaForceRedisplay();
   }
   return super.handleEvent(e);
}

public boolean action(Event e, Object arg) {
   JotsaAnimationObject tempanim;
   JotsaAnimationObject master;
   JotsaAnimationCollection group;
   JotsaAnimationCollection col;
   JotsaAnimationCollection col1;
   JotsaAnimationCollection col2;
   Color old_color;
   int dim[];
   double sval;
   int init_x;
   int init_y;
   if ("Select G1".equals(arg)) {
      System.out.println("Select G1");
      showStatus("Select G1");
      select_group_1_flag = true;
      select_group_2_flag = false;
      drag_group_flag = false;
      if (select1 != null)
         Collections.hide_master(select1);
      selector_2.set_inhibit_display();
      selector_d.set_inhibit_display();
      selector_1.set_position(mouse_x,mouse_y);
      selector_1.clear_inhibit_display();
      return true;
   }
   if ("Select G2".equals(arg)) {
      System.out.println("Select G2");
```

```
         showStatus("Select G2");
         select_group_2_flag = true;
         select_group_1_flag = false;
         drag_group_flag = false;
         if (select2 != null)
            Collections.hide_master(select2);
         selector_1.set_inhibit_display();
         selector_d.set_inhibit_display();
         selector_2.set_position(mouse_x,mouse_y);
         selector_2.clear_inhibit_display();
         return true;
      }
      if ("Drag Group".equals(arg)) {
         System.out.println("Drag Group");
         showStatus("Drag Group");
         drag_group_flag = true;
         select_group_1_flag = false;
         select_group_2_flag = false;
         selector_1.set_inhibit_display();
         selector_2.set_inhibit_display();
         selector_d.set_position(mouse_x,mouse_y);
         selector_d.clear_inhibit_display();
         return true;
      }
      if ("Make Group".equals(arg)) {
         System.out.println("Make Group");
         master = Collections.create_master_object(200,100,Color.green);
         group = new JotsaAnimationCollection(master);
         tempanim = Collections.create_split_object();
         group.insert(tempanim,0,0);
         Collections.add_collection(group);
         return true;
      }
      if ("Split G1".equals(arg)) {
         System.out.println("Split G1");
         if (select1 == null) return true;
         Collections.hide_master(select1);
         col = select1;
         select1 = null;
         new Split(this,Collections,col,speed_control.value());
         return true;
      }
      if ("Merge G1 G2".equals(arg)) {
         System.out.println("Merge G1 G2");
         if (select1 == null) return true;
         if (select2 == null) return true;
         Collections.hide_master(select1);
         Collections.hide_master(select2);
         col1 = select1;
         col2 = select2;
         select1 = null;
         select2 = null;
         new Merge(this,Collections,col1,col2,speed_control.value());
         return true;
      }
      if ("Show Collection".equals(arg)) {
         System.out.println("Show Collection");
         Collections.show_clist();
      }
      if ("Show Disp".equals(arg)) {
         System.out.println("Show Disp");
         show_disp();
      }
      return super.action(e,arg);
   }
}
```

References

[1] Steven Robbins, “A JOTSA Example,” UTSA Computer Science Technical Report, 1997