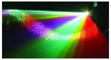
Chapter 9
Described Dynamic Visual Content

The Design and Implementation of Multimedia Software

David Bernstein

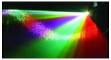
Jones and Bartlett Publishers

www.jbpub.com



About this Chapter

- This chapter considers ways in which one can describe the way the visual ‘stream’ changes over time.
- This chapter uses the analogy of the theater (and acting).

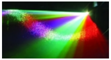


Requirements

F9.1 Manage a collection of sprites.

F9.2 Repeatedly inform each sprite that it should perform the next task in its script.

F9.3 Render the sprites.



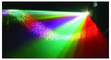
Alternative 1

- Approach:

  Add code to the `Visualization` class.

- Shortcomings:

  What are the shortcomings?



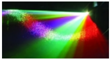
Alternative 1

- Approach:

  Add code to the `Visualization` class.

- Shortcomings:

  Complexity – There is no reason that someone who is interested in static visual content should have to understand features that are required to work with dynamic visual content.



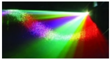
Alternative 2

- Approach:

  Use the decorator pattern.

- Shortcomings:

  What are the shortcomings?



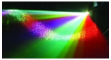
Alternative 2

- Approach:

  Use the decorator pattern.

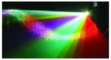
- Shortcomings:

  It is hard to imagine a situation in which, at run time, one would want to add these kinds of capabilities to a `Visualization` object.



Alternative 3

Create a `Stage` class that specializes the `Visualization` class.



Comparison to the `Screen` Class

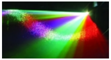
- Similarities:

  Addition of a `Metronome`.

  The ability to respond to ‘ticks’.

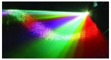
- Differences:

  No notion of a frame (since sprites might need to change their behavior at any time).



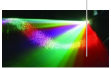
Sprite

```
package visual.dynamic.described;

public interface Sprite extends event.MetronomeListener,
                                visual.statik.TransformableContent
{
}
```



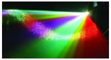
Stage – Structure

```
package visual.dynamic.described;

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

import event.*;
import visual.*;
import visual.statik.sampled.*;

public class Stage extends    Visualization
                   implements MetronomeListener
{
    private boolean              shouldRestart;
    private int                  timeStep, restartTime, time;
    private Metronome            metronome;

    public Stage(int timeStep)
    {
       this(timeStep, new Metronome(timeStep));
    }

    public Stage(int timeStep, Metronome metronome)
    {
       super();

       this.timeStep     =  timeStep;
       time              = -timeStep;
       shouldRestart     =  false;
```



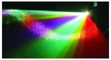
Stage – Structure (cont.)

```
        restartTime       = -1;
        this.metronome    = metronome;
        setBackground(Color.WHITE);

        // The first listener is notified last
        metronome.addListener(this);
    }
}
```



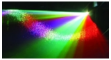
Stage – Metronome

```
    public void setRestartTime(int restartTime)
    {
       if (restartTime < 0)
       {
          this.restartTime = -1;
          shouldRestart = false;
       }
       else
       {
          this.restartTime = restartTime;
          shouldRestart = true;
       }
    }

    public void start()
    {
       metronome.start();
    }

    public void stop()
    {
       metronome.stop();
    }
```



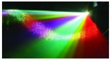
Stage – Managing Sprite Objects

```
    public void add(Sprite sprite)
    {
       // Make the Sprite a MetronomeListener
       metronome.addListener(sprite);

       // Treat the Sprite as a SimpleContent and
       // add it to the Visualization
       super.add(sprite);
    }

    public void remove(Sprite sprite)
    {
       metronome.removeListener(sprite);
       super.remove(sprite);
    }
```



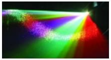
Stage – handleTick()

```
public void handleTick(int time)
{
   if ((shouldRestart) && (time > restartTime))
   {
      metronome.setTime(-timeStep);
   }

   repaint();
}
```

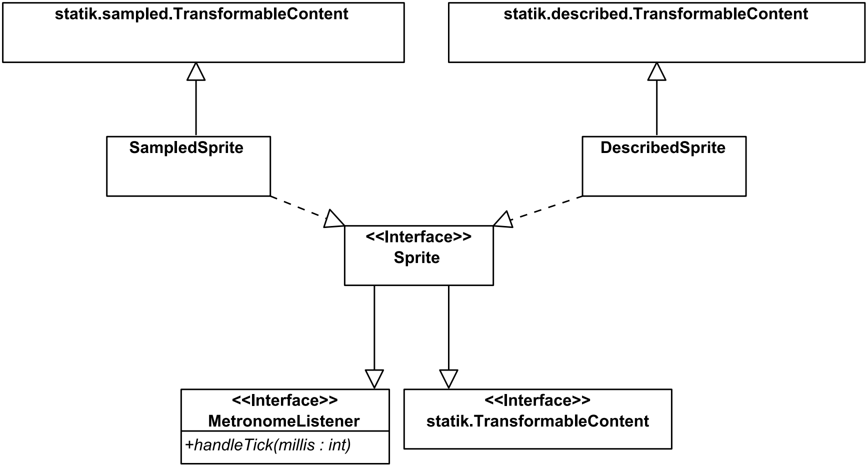


Satisfying Requirements 9.2 and 9.3

- A class that implements the `Sprite` interface must have a `handleTick()`.
- A class that implements the `Sprite` interface must have a `render()` method.



Alternative 1



statik.sampled.TransformableContent

statik.described.TransformableContent

SampledSprite

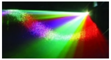
DescribedSprite

<<Interface>>
Sprite

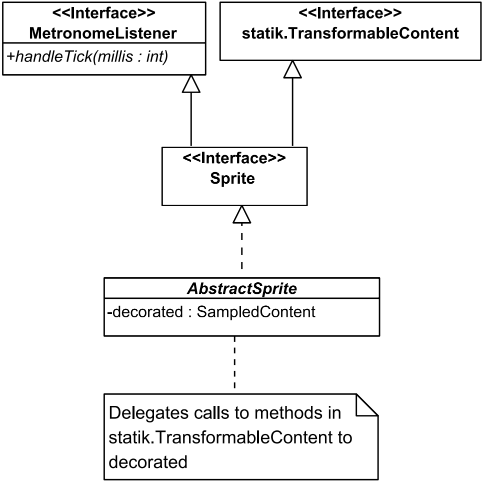
<<Interface>>
MetronomeListener

+handleTick(millis : int)

<<Interface>>
statik.TransformableContent



Alternative 2



<<Interface>>
MetronomeListener
+handleTick(millis : int)

<<Interface>>
statik.TransformableContent

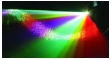
<<Interface>>
Sprite

AbstractSprite
-decorated : SampledContent

Delegates calls to methods in statik.TransformableContent to decorated

Advantages of Alternative 2

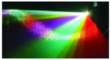
What are the advantages?



Advantages of Alternative 2

Can decorate different `SimpleContent` objects in the same way (e.g., `FallingSprite` could decorate `SimpleContent` that looks like a leaf, a raindrop, a snowflake, etc).

Can associate a different `SimpleContent` object with a particular `Sprite` at different points in time (e.g., a `WalkingPersonSprite` might use different `sampled.Content` objects to represent its legs at different points in the walking process).



The Next Question to Address

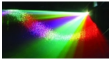
- The Question:

  How to incorporate a ‘script’ in objects that implement the `Sprite` interface.

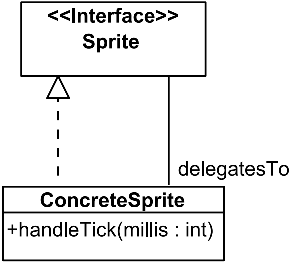
- Common Approaches:

  Use ‘rules’

  Interpolate between known states



Alternative 1



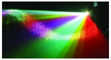
<<Interface>>
Sprite

delegatesTo

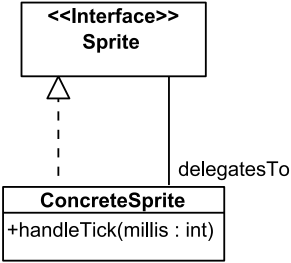
ConcreteSprite

+handleTick(millis : int)

What are the shortcomings?

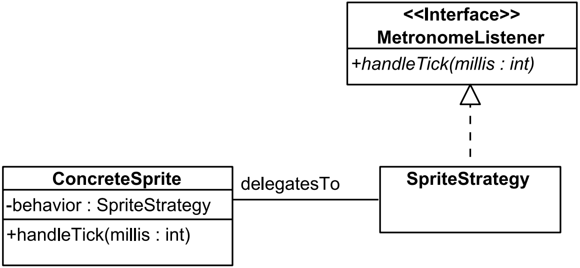


Alternative 1



It is a little confusing since a `ConcreteSprite` decorates an `AbstractSprite` which, in turn, decorates a `TransformableContent` object.

Alternative 2



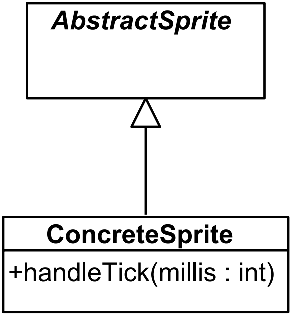
<<Interface>>
MetronomeListener
+handleTick(millis : int)

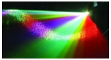
ConcreteSprite
-behavior : SpriteStrategy
+handleTick(millis : int)

delegatesTo

SpriteStrategy

Alternative 3





Comparing Alternatives 2 and 3

- Thoughts:

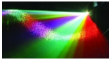
  Both have a lot to offer.

  Specialization is simpler.

  It seems unlikely that the system will need to change a rule-based sprite to an interpolating sprite at run-time.

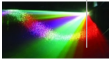
- Fortunately:

  One could use the strategy pattern in the future without breaking any ‘legacy’ classes that used specialization.



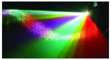
AbstractSprite – Structure

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;

import visual.statik.TransformableContent;

public abstract class      AbstractSprite
                implements Sprite
{
    protected boolean     rotationPoint, visible;
    protected double      angle, rotationX, rotationY;
    protected double      scaleX, scaleY, x, y;

    public AbstractSprite()
    {
       super();

       x      = 0.0;
       y      = 0.0;
       angle  = 0.0;
       scaleX = 1.0;
       scaleY = 1.0;

       rotationPoint = false;
       rotationX     = 0.0;
```



AbstractSprite – Structure (cont.)

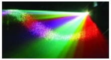
```
       rotationY     = 0.0;
    }
}
```



`AbstractSprite` – Some Abstract Methods

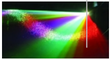
```
    public abstract void handleTick(int time);
```

```
    protected abstract TransformableContent getContent();
```



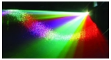
AbstractSprite – Setters

```
   public void setLocation(double x, double y)
   {
      this.x = x;
      this.y = y;
   }

   public void setRotation(double r, double x, double y)
   {
      rotationPoint = true;
      this.angle    = r;
      this.x        = x;
      this.y        = y;
   }

   public void setRotation(double r)
   {
      rotationPoint = false;
      this.angle    = r;
   }

   public void setScale(double sx, double sy)
   {
      scaleX = sx;
      scaleY = sy;
   }

   public void setScale(double s)
   {
```



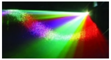
AbstractSprite – Setters (cont.)

```
       setScale(s, s);
    }

    public void setVisible(boolean v)
    {
        visible = v;
    }
}
```



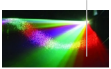
AbstractSprite – getBounds()

```
   public Rectangle2D getBounds2D(boolean ofTransformed)
   {
      return getContent().getBounds2D(ofTransformed);
   }

   public Rectangle2D getBounds2D()
   {
      return getBounds2D(true);
   }
```



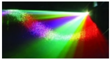
AbstractSprite – Rendering

```
public void render(Graphics g)
{
   double                   rx, ry;
   Rectangle2D              bounds;
   TransformableContent     tc;

   if (visible)
   {
      tc = getContent();

      if (tc != null)
      {
         // Find the point to rotate around
         if (rotationPoint)
         {
            rx = rotationX;
            ry = rotationY;
         }
         else
         {
            bounds = tc.getBounds2D(false);
            rx     = bounds.getWidth()/2.0;
            ry     = bounds.getHeight()/2.0;
         }

         // Transform
         tc.setLocation(x, y);
         tc.setRotation(angle, rx, ry);
```



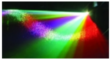
AbstractSprite – Rendering (cont.)

```
         tc.setScale(scaleX, scaleY);

         // Render
         tc.render(g);
      }
   }
}
```



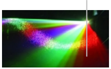
What’s Next?

- We need to create specializations of the `AbstractSprite` class.
- For example, let’s consider a simple rule-based sprite that ‘floats’ from the top of the `Stage` to the bottom of the `Stage`.



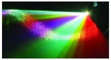
FloatingSprite – Structure

```
import java.awt.*;
import java.awt.geom.*;
import java.util.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class FloatingSprite extends AbstractSprite
{
    private double                    maxX, maxY, x, y;
    private Random                    rng;
    private TransformableContent      content;

    public FloatingSprite(TransformableContent content,
                          double width, double height)
    {
       super();
       this.content = content;
       maxX         = width;
       maxY         = height;

       rng = new Random();

       x = rng.nextDouble()*maxX;
       y = 0.0;
       setLocation(x, y);

       setVisible(true);
```



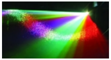
FloatingSprite – Structure (cont.)

```
    }
}
```



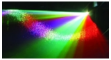
FloatingSprite – getContent()

```
public TransformableContent getContent()
{
   return content;
}
```



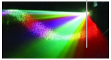
FloatingSprite – handleTick()

```
   public void handleTick(int time)
   {
      double        n;

      n = rng.nextDouble();
      if      (n < 0.80)   y += 2.0;
      else if (n > 0.90)   y -= 1.0;

      n = rng.nextDouble();
      if      (n < 0.20) x -= 1.0;
      else if (n > 0.80) x += 1.0;

      // Check if at the bottom
      if (y > maxY)
      {
         y  = 0.0;
         x = rng.nextDouble()*maxX;
      }

      setLocation(x, y);
   }
```



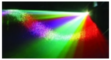
FloatingSpriteApp

```
        ContentFactory                  factory;
        FloatingSprite                  sprite;
        int                             height, width;
        JPanel                          contentPane;
        ResourceFinder                  finder;
        Stage                           stage;
        TransformableContent            content;
        VisualizationView               stageView;

        width   = 640;
        height  = 480;

        finder  = ResourceFinder.createInstance(this);
        factory = new ContentFactory(finder);

        // The Stage
        stage        = new Stage(50);
        stage.setBackground(new Color(255, 255, 255));
        stageView = stage.getView();
        stageView.setBounds(0,0,width,height);

        // The Sprite
        content = factory.createContent("snowflake.png", 4, false);
        sprite  = new FloatingSprite(content, width, height);
        stage.add(sprite);

        // The content pane
        contentPane = (JPanel)rootPaneContainer.getContentPane();
```

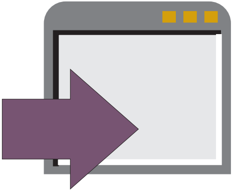


FloatingSpriteApp (cont.)

```
        contentPane.add(stageView);

        // Start the dynamics
        stage.start();
```



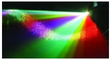
FloatingSpriteApp – Demonstration



In `examples/chapter`:

FloatingSprite.html

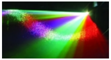
```
java -cp FloatingSprite.jar FloatingSpriteApplication
```



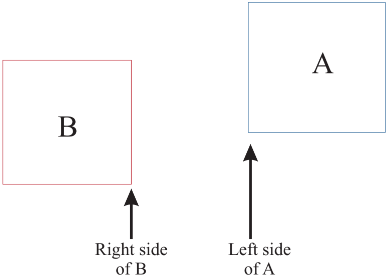
Requirements

F9.4 Allow one sprite to interact with another.

F9.5 Allow the user to interact with sprites.



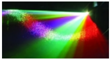
Determining if Rectangles Do Not Intersect



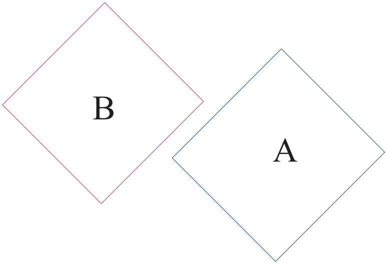
Determining if Rectangles Do Not Intersect (cont.)

Letting `rightA`, `leftA`, `topA` and `botA` denote the right, left, top, and bottom of A, and `rightB`, `leftB`, `topB` and `botB` denote the right, left, top, and bottom of B, the expression to use to test if A and B **do not** intersect is:

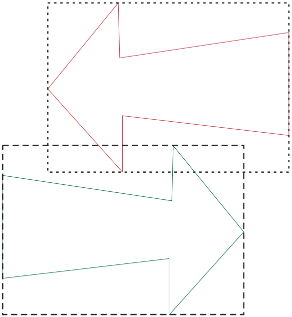
```
(rightA < leftB) || (leftA > rightB) || (botA < topB) || (topA > botB)
```



Intersection of Non-Rectangular, Convex Sprites

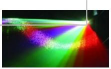


Using Bounding Boxes



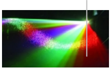
AbstractSprite – intersects()

```
    public boolean intersects(Sprite s)
    {
        boolean          retval;
        double           maxx, maxy, minx, miny;
        double           maxx0, maxy0, minx0, miny0;
        Rectangle2D      r;

        retval = true;

        r = getBounds2D(true);
        minx = r.getX();
        miny = r.getY();
        maxx = minx + r.getWidth();
        maxy = miny + r.getHeight();

        r = s.getBounds2D(true);
        minx0 = r.getX();
        miny0 = r.getY();
        maxx0 = minx0 + r.getWidth();
        maxy0 = miny0 + r.getHeight();

        if ( (maxx < minx0) || (minx > maxx0) ||
             (maxy < miny0) || (miny > maxy0) ) retval = false;

        return retval;
    }
```



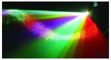
RuleBasedSprite

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;
import java.util.*;

import visual.statik.TransformableContent;

public abstract class RuleBasedSprite extends AbstractSprite
{
    protected ArrayList<Sprite>         antagonists;
    protected TransformableContent      content;

    public RuleBasedSprite(TransformableContent content)
    {
       super();

       antagonists = new ArrayList<Sprite>();
       this.content = content;
       setVisible(true);
    }

    public void addAntagonist(Sprite antagonist)
    {
       antagonists.add(antagonist);
    }

    public TransformableContent getContent()
```



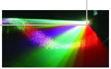
RuleBasedSprite (cont.)

```
    {
       return content;
    }

    public abstract void handleTick(int time);


    public void removeAntagonist(Sprite antagonist)
    {
       antagonists.remove(antagonist);
    }
}
```



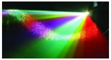
Fish – Structure

```
import java.util.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class Fish extends RuleBasedSprite
{
    protected double      initialSpeed, maxX, maxY, speed, x, y;

    private static final int      INITIAL_LOCATION = -320;
    private static final Random  rng = new Random();

    public Fish(TransformableContent content,
                double width, double height, double speed)
    {
       super(content);
       maxX = width;
       maxY = height;

       x    = rng.nextDouble()*maxX;
       y    = rng.nextInt()*maxY;

       this.initialSpeed = speed;
       this.speed        = speed;
    }
}
```



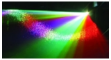
Fish – updateLocation()

```
protected void updateLocation()
{
   x += speed;

   if (x > (int)maxX)
   {
      x     = INITIAL_LOCATION;
      y     = rng.nextDouble()*maxX;
      speed = initialSpeed;
   }

   // Set the location
   setLocation(x, y);
}
```



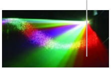
Fish – handleTick()

```
public void handleTick(int time)
{
   double             initialSpeed;
   Iterator<Sprite>   i;
   Sprite             shark;

   initialSpeed = speed;

   i = antagonists.iterator();
   while (i.hasNext())
   {
      shark = i.next();
      if (intersects(shark)) speed = 20.;
   }

   updateLocation();
}
```



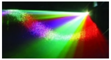
FishTankApp

```
        width  = 640;
        height = 480;

        finder       = ResourceFinder.createInstance(this);
        factory      = new ContentFactory(finder);
        imageFactory = new ImageFactory(finder);

        // The Stage
        stage = new Stage(50);
        stage.setBackground(Color.blue);
        content = factory.createContent("ocean.png", 3, false);
        stage.add(content);
        stageView = stage.getView();
        stageView.setBounds(0,0,width,height);

        // The Shark
        content = factory.createContent("shark.png", 4, false);
        shark = new Fish(content, width, height, 8.);
        stage.add(shark);

        // The school of Fish
        // (Use the same BufferedImage object for all Fish)
        image   = imageFactory.createBufferedImage("fish.png", 4);
        for (int i=0; i<10; i++)
        {
           content = factory.createContent(image, false);
           fish = new Fish(content, width, height, 3.);
           fish.addAntagonist(shark);
```



FishTankApp (cont.)

```
   stage.add(fish);
}

// The content pane
contentPane = (JPanel)rootPaneContainer.getContentPane();
contentPane.add(stageView);

stage.start();
```

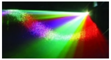


`FishTankApp` – Demonstration

In `examples/chapter`:

FishTank.html

```
java -cp FishTank.jar FishTankApplication
```



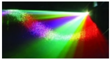
Sprites with Multiple Pieces of Content

- The Objective:

  Make a fish appear to move it's tail.

- What's Needed?:

  What's Needed?



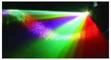
Sprites with Multiple Pieces of Content

- The Objective:

  Make a fish appear to move it's tail.

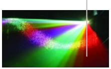
- What's Needed?:

  Different `Content` objects for different states.



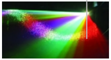
Sprites with Multiple Pieces of Content (cont.)

```
import java.util.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class SwimmingFish extends RuleBasedSprite
{
    protected double      initialSpeed, maxX, maxY, speed, x, y;
    protected int         lastTime, millisPerState, state, stateChange;
    protected int         timeInState;
    protected TransformableContent[]  contents;



    private static final int     INITIAL_LOCATION = -320;
    private static final Random  rng = new Random();

    public SwimmingFish(TransformableContent[] contents,
                        double width, double height, double speed)
    {
       super(contents[0]);

       this.contents = contents;

       maxX = width;
       maxY = height;

       x    = rng.nextDouble()*maxX;
```



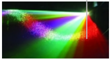
Sprites with Multiple Pieces of Content (cont.) (cont.)

```
      y    = rng.nextInt()*maxY;

      this.initialSpeed = speed;
      this.speed        = speed;
      state             = 0;
      lastTime          = 0;
      timeInState       = 0;
      stateChange       = 1;
   }


   public TransformableContent getContent()
   {
      return contents[state];
   }


   public void handleTick(int time)
   {
      double            initialSpeed;
      Iterator<Sprite>  i;
      Sprite            shark;

      initialSpeed = speed;

      i = antagonists.iterator();
      while (i.hasNext())
      {
```



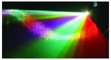
Sprites with Multiple Pieces of Content (cont.) (cont.)

```
         shark = i.next();
         if (intersects(shark)) speed = 20.;
      }

      millisPerState    = 500 - (int)(speed*20);

      timeInState += (time - lastTime);
      if (timeInState > millisPerState)
      {
         timeInState = 0;
         state += stateChange;
         if      (state == 2) stateChange = -1;
         else if (state == 0) stateChange =  1;
      }
      lastTime = time;

      updateLocation();
   }


   protected void updateLocation()
   {
      x += speed;

      if (x > (int)maxX)
      {
         x     = INITIAL_LOCATION;
         y     = rng.nextDouble()*maxX;
```



Sprites with Multiple Pieces of Content (cont.) (cont.)

```
            speed = initialSpeed;
        }

        // Set the location
        setLocation(x, y);
    }
}
```



SwimmingFishTankApp – Demonstration

In extras:

SwimmingFishTank.html

```
java -cp SwimmingFishTank.jar SwimmingFishTankApplication
```

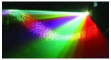
Sprites that Move Together

- The Objective:

  Different ¡code¿Sprite¡/code¿ objects move together (e.g., exhaust coming out of a bus).

- What's Needed?:

  What's Needed?



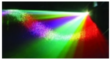
Sprites that Move Together

- The Objective:

  Different ¡code¿Sprite¡/code¿ objects move together (e.g., exhaust coming out of a bus).

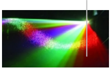
- What's Needed?:

  One `Sprite` object that "controls" other `Sprite` objects.



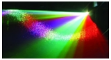
Sprites that Move Together (cont.)

```
import java.awt.*;
import java.awt.geom.*;

import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class BigBus extends RuleBasedSprite
{
    private double              maxX, maxY, x, y;
    private Exhaust[]           exhaust;

    public BigBus(TransformableContent content,
                  double width, double height,
                  Stage stage)
    {
       super(content);

       exhaust = new Exhaust[15];
       for (int i=0; i<exhaust.length; i++)
       {
          exhaust[i] = new Exhaust();
          stage.add(exhaust[i]);
       }

       x    =   0.0;
       y    = 300.0;
       maxX = width;
       maxY = height;
```



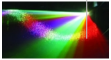
Sprites that Move Together (cont.) (cont.)

```
    }

    public void handleTick(int millis)
    {
       // Move the bus
       x = x + 1;
       setLocation(x, y);
       if (x > maxX+50)
       {
          setVisible(false);
          for (int i=0; i<exhaust.length; i++)
             exhaust[i].setVisible(false);
       }

       // Set the origin for the Exhaust objects
       for (int i=0; i<exhaust.length; i++)
          exhaust[i].setOrigin(x, y+63);

       // Inform the Exhaust objects that a tick has occurred
       for (int i=0; i<exhaust.length; i++)
          exhaust[i].handleTick(millis);
    }
}
```



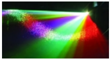
Sprites that Move Together (cont.) (cont.)

```
import java.awt.*;
import java.awt.geom.*;
import java.util.Random;

import visual.dynamic.described.*;
import visual.statik.described.*;

public class Exhaust extends RuleBasedSprite
{
    private double        originX, originY;
    private int           count, length, slope;

    private static final  int      DIAMETER = 5;
    private static final  Random   rng      = new Random();

    public Exhaust()
    {
       super(new Content(new Ellipse2D.Float(0,0,DIAMETER,DIAMETER),
                         Color.BLACK,
                         Color.GRAY,
                         new BasicStroke()
                         )
             );

       length = rng.nextInt(15);
       count  = -1;
    }
```

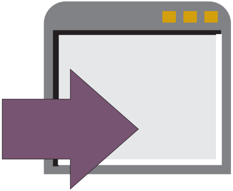


Sprites that Move Together (cont.) (cont.)

```
    public void handleTick(int millis)
    {
       count++;

       if (count >= length)
       {
          count = 0;
          setLocation(originX, originY);
       }
       else
       {
          slope  = rng.nextInt(4) - 1;
          setLocation(originX-count, originY-(count*slope));
       }
    }

    public void setOrigin(double x, double y)
    {
       originX = x - DIAMETER/2;
       originY = y - DIAMETER/2;
    }
}
```



Sprites that Move Together – Demonstration



In extras:

BigBus.html

```
java -cp BigBus.jar BigBusApplication
```

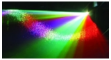
Satisfying Requirement 9.5

- Interested `Sprite` objects must be able to observe user-generated events.

  They can implement the `KeyListener` interface and/or the `MouseListener` and `MouseMotionListener` interfaces.

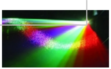
- A subject is needed.

  The `VisualizationView` class extends the `JComponent` class, and the `JComponent` class provides this functionality.



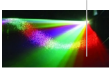
Visualization – Key Listeners

```
public void addKeyListener(KeyListener kl)
{
   Iterator<VisualizationView>  i;
   VisualizationView            view;

   i = getViews();
   while (i.hasNext())
   {
      view = i.next();
      view.addKeyListener(kl);
   }
}

public synchronized void removeKeyListener(
                               KeyListener kl)
{
   Iterator<VisualizationView>  i;
   VisualizationView            view;

   i = getViews();
   while (i.hasNext())
   {
      view = i.next();
      view.removeKeyListener(kl);
   }
}
```



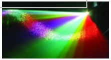
Visualization – Mouse Listeners

```
    public void addMouseListener(MouseListener ml)
    {
       Iterator<VisualizationView>  i;
       VisualizationView            view;

       i = getViews();
       while (i.hasNext())
       {
          view = i.next();
          view.addMouseListener(ml);
       }
    }

    public void addMouseMotionListener(
                        MouseMotionListener mml)
    {
       Iterator<VisualizationView>  i;
       VisualizationView            view;

       i = getViews();
       while (i.hasNext())
       {
          view = i.next();
          view.addMouseMotionListener(mml);
       }
    }

    public synchronized void removeMouseListener(
```



Visualization – Mouse Listeners (cont.)

```
                                  MouseListener ml)
   {
      Iterator<VisualizationView>  i;
      VisualizationView            view;

      i = getViews();
      while (i.hasNext())
      {
         view = i.next();
         view.removeMouseListener(ml);
      }
   }

   public synchronized void removeMouseMotionListener(
                                  MouseMotionListener mml)
   {
      Iterator<VisualizationView>  i;
      VisualizationView            view;

      i = getViews();
      while (i.hasNext())
      {
         view = i.next();
         view.removeMouseMotionListener(mml);
      }
   }
```



An Example

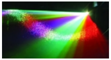
- The Setting:

  An amazingly addictive (and/or unbearably stupid) balloon popping game.

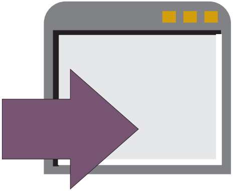
- The Participants:

  `Cupola`

  `Balloon`



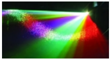
An Example – Demonstration



In `examples/chapter`:

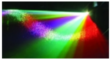
Balloon.html

```
java -cp Balloon.jar BalloonApplication
```



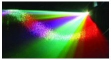
Cupola – Structure

```
import java.awt.event.*;
import java.awt.geom.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class Cupola extends    RuleBasedSprite
                    implements MouseMotionListener
{
    private double     left, top;

    public Cupola(TransformableContent content,
                  double stageWidth, double stageHeight)
    {
       super(content);
       Rectangle2D        bounds;

       bounds = content.getBounds2D(false);
       top    = (stageHeight - bounds.getHeight() - 34);
       left   = (stageWidth  - bounds.getWidth())/2.0;

       setLocation(left, top);
    }
}
```



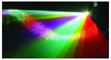
Cupola – MouseMotionListener

```
    public void mouseDragged(MouseEvent evt)
    {
       mouseMoved(evt);
    }

    public void mouseMoved(MouseEvent evt)
    {
       this.left = (double)evt.getX();
    }
```



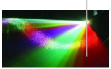
Cupola – handleTick()

```
    public void handleTick(int time)
    {
       setLocation(left, top);
    }
```



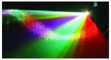
Balloon – handleTick()

```
public void handleTick(int time)
{
   Sprite   cupola;

   // Wait for the initial rendering
   if (time < 100) return;

   // Check for an intersection
   cupola = null;
   if (antagonists.size() > 0) cupola = antagonists.get(0);


   if ((cupola != null) && (intersects(cupola)))
   {
      speed = 0;
      setVisible(false);
   }

   // Update the location
   top += speed;

   if (top > maxY)
   {
      left   = rng.nextInt(maxX);
      top    = minY;
      speed  = rng.nextInt(10);
   }
```



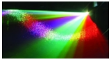
Balloon – handleTick() (cont.)

```
        // Set the location
        setLocation(left, top);
    }
```



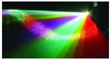
Jobs in Traditional Cel Animation

- Drawing backgrounds.
- Drawing key/important frames.
- Drawing all of the frames in between the key frames.



Requirements

F9.6 Support the description of dynamic behavior using key-times and tweening.



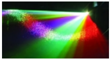
Alternative 1

- Approach:

  Store the attributes of the `TransformableContent` objects at each key time in the `TransformableContent` objects themselves.

- Shortcomings:

  What are the shortcomings?



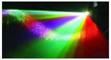
Alternative 1

- Approach:

  Store the attributes of the `TransformableContent` objects at each key time in the `TransformableContent` objects themselves.

- Shortcomings:

  It makes it difficult to interpolate between the key times.



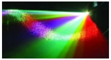
Alternative 2

- Approach:

  Keep the attributes for each of the key times external to the `TransformableContent` objects.

- Advantages:

  What are the advantages?



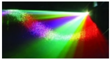
Alternative 2

- Approach:

  Keep the attributes for each of the key times external to the `TransformableContent` objects.

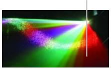
- Advantages:

  The `Sprite` has easy access to all of the information it needs to calculate the attributes at the in-between times.



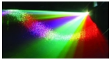
TweeningSprite – Structure

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;

public abstract class TweeningSprite extends AbstractSprite
{
    private   double           frac;
    private   int              currentIndex, endState, lastTime;
    private   int              nextIndex, nextKT;
    protected Vector<Integer>  keyTimes;
    protected Vector<Point2D>  locations;
    protected Vector<Double>   rotations, scalings;

    public static final int REMAIN         = 0;
    public static final int REMOVE         = 1;

    public TweeningSprite()
    {
       super();

       keyTimes  = new Vector<Integer>();
       locations = new Vector<Point2D>();
       rotations = new Vector<Double>();
       scalings  = new Vector<Double>();
       endState  = REMAIN;
```



TweeningSprite – Structure (cont.)

```
       initialize();
    }
}
```

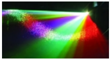


TweeningSprite – addKeyTime()

```
protected int addKeyTime(int keyTime, Point2D location,
                         Double rotation, Double scaling)
{
   boolean     keepLooking;
   int         existingKT, i, index;

   existingKT  = -1;
   keepLooking = true;

   i = 0;
   while ((i < keyTimes.size()) && keepLooking)
   {
      existingKT = ((Integer)keyTimes.get(i)).intValue();

      if (existingKT >= keyTime) keepLooking = false;
      else                       i++;
   }

   if ((existingKT == i) && !keepLooking)  // Duplicate
   {
      i = -1;
   }
   else
   {
      keyTimes.insertElementAt(new Integer(keyTime), i);
      locations.insertElementAt(location, i);
      rotations.insertElementAt(rotation, i);
      scalings.insertElementAt(scaling, i);
```

TweeningSprite – addKeyTime() (cont.)

```
       }

       return i;
    }
```

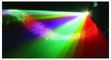


Linear Interpolation

Letting a_t denote the value of the attribute at the previous/current key time and a_{t+1} denote the value of the attribute at the next key time, the in-between value, $b(\lambda)$, is then given by:

$$b(\lambda) = (1 - \lambda)a_t + \lambda a_{t+1} \tag{1}$$

where $\lambda \in [0, 1]$ denotes the interpolation fraction.

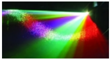


Linear Interpolation (cont.)

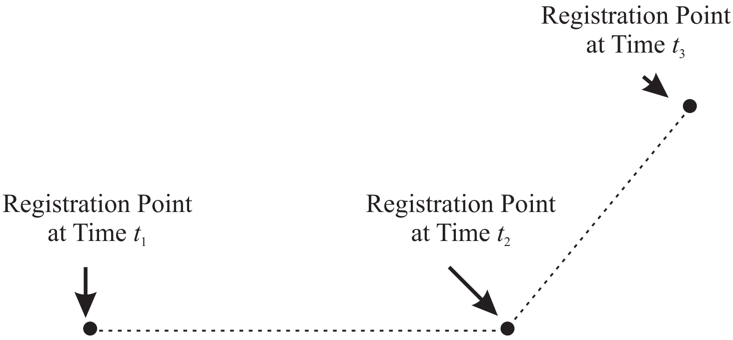
Note that (1) implies:

$$\begin{aligned} b(\lambda) &= a_t - \lambda a_t + \lambda a_{t+1} && (2) \\ &= a_t + \lambda(a_{t+1} - a_t) && (3) \end{aligned}$$

which is the more widely-used form.



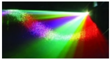
Location Tweening



Registration Point
at Time t_3

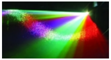
Registration Point
at Time t_1

Registration Point
at Time t_2

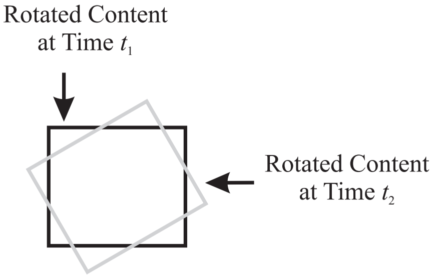


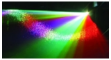
TweeningSprite – tweenLocation

```
protected void tweenLocation(int currentIndex, int nextIndex,
                             double frac)
{
   double          x, y;
   Point2D         currentKTLocation, nextKTLocation;

   currentKTLocation = locations.get(currentIndex);
   nextKTLocation    = locations.get(nextIndex);

   x = currentKTLocation.getX() +
       frac*(nextKTLocation.getX()- currentKTLocation.getX());
   y = currentKTLocation.getY() +
      frac*(nextKTLocation.getY() - currentKTLocation.getY());

   setLocation(x, y);
}
```



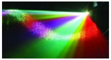
Pure Rotation Tweening



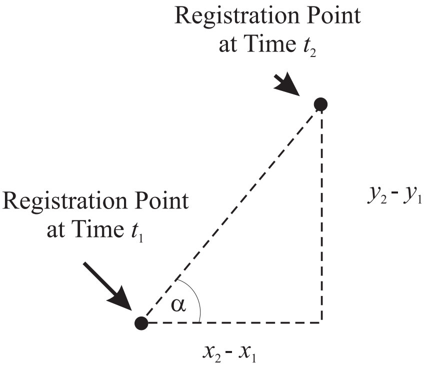


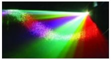
`TweeningSprite` – Pure Rotation Tweening

```
       currentKTRotation = rotation.doubleValue();

       rotation = rotations.get(nextIndex);
       if (rotation == null) nextKTRotation = currentKTRotation;
       else                  nextKTRotation = rotation.doubleValue();

       r = currentKTRotation + frac*(nextKTRotation-currentKTRotation);
    }
```



Aligned Rotation Tweening





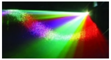
Pure Rotation Tweening (cont.)

Thinking of the current segment as the hypotenuse of a right triangle, the difference in y values defines the length of the side opposite the angle of interest (denoted by α), and the difference in x values defines the length of the adjacent side. Hence:

$$\tan(\alpha) = \frac{y_2 - y_1}{x_2 - x_1} \tag{4}$$

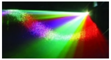
It follows that:

$$\alpha = \text{atan}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{5}$$



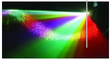
TweeningSprite – Aligned Rotation Tweening

```
        currentKTLocation = locations.get(currentIndex);
        nextKTLocation    = locations.get(nextIndex);

        r=Math.atan((nextKTLocation.getY()-currentKTLocation.getY())/
                    (nextKTLocation.getX()-currentKTLocation.getX()) );
```



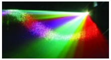
An Example: `Airplane`

```
import java.awt.geom.*;
import java.awt.image.*;

import io.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class Airplane extends SampledSprite
{
    public Airplane()
    {
        super();
        Content            content;
        ContentFactory     factory;

        factory = new ContentFactory(ResourceFinder.createInstance(this));
        content = factory.createContent("airplane.png", 4);
        addKeyTime(  500,   0.0, 350.0, -0.75, 1.0, content);
        addKeyTime( 2000, 100.0, 200.0, -0.30, 1.0, null);
        addKeyTime( 4000, 200.0,  50.0,  0.00, 1.0, null);
        addKeyTime( 6000, 300.0,  50.0,  0.20, 1.0, null);
        addKeyTime( 8000, 400.0, 200.0,  0.00, 1.0, null);
        addKeyTime( 8500, 500.0, 200.0,  0.00, 1.0, null);
        setEndState(REMOVE);
    }

    private void addKeyTime(int time, double x, double y,
                             double r, double s, Content c)
```

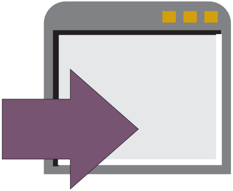


An Example: `Airplane` (cont.)

```
    {
       addKeyTime(time, new Point2D.Double(x, y), new Double(r),
                   new Double(s), c);
    }
}
```



Airplane – Demonstration



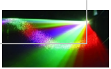
In `examples/chapter`:

Airplane.html

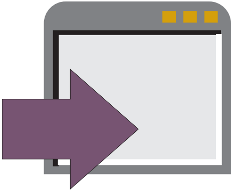
```
java -cp Airplane.jar AirplaneApplication
```

Another Example: `BuzzyOnMars`

```
import java.awt.geom.Point2D;

import visual.dynamic.described.DescribedSprite;
import visual.statik.described.*;

public class BuzzyOnMars extends DescribedSprite
{
    public BuzzyOnMars()
    {
        super();
        BuzzyStanding       buzzy;

        buzzy = new BuzzyStanding();
        addKeyTime(  500,   0.0, 380.0,  0.00, 1.0, buzzy);
        addKeyTime( 2000, 180.0, 380.0,  0.00, 1.0, null);
        addKeyTime( 4000, 180.0,  75.0,  0.20, 1.0, null);
        addKeyTime( 6000, 640.0,  20.0,  6.48, 1.0, null);
        setEndState(REMOVE);
    }

    private void addKeyTime(int time, double x, double y,
                             double r, double s, AggregateContent c)
    {
       addKeyTime(time, new Point2D.Double(x, y), new Double(r),
                   new Double(s), c);
    }
}
```



`BuzzyOnMars` – Demonstration



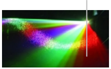
In `examples/chapter`:

BuzzyOnMars.html

```
java -cp BuzzyOnMars.jar BuzzyOnMarsApplication
```

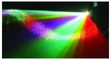
A Last Example: `BusOnRoute`

```
import java.awt.geom.Point2D;

import io.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class BusOnRoute extends SampledSprite
{
    public BusOnRoute()
    {
       super();
       Content                content;
       ContentFactory         factory;
       ResourceFinder         finder;

       finder  = ResourceFinder.createInstance(this);
       factory = new ContentFactory(finder);
       content = factory.createContent("bus.png");
       addKeyTime( 0, 164, 210, content);
       addKeyTime( 1, 310, 255, null);
       addKeyTime( 2, 314, 234, null);
       addKeyTime( 3, 401, 231, null);
       addKeyTime( 4, 419, 269, null);
       addKeyTime( 5, 353, 340, null);
       addKeyTime( 6, 430, 367, null);
       addKeyTime( 7, 420, 418, null);
       addKeyTime( 8, 450, 421, null);
       addKeyTime( 9, 454, 386, null);
```



A Last Example: `BusOnRoute` (cont.)

```
       addKeyTime(10, 512, 393, null);
       addKeyTime(11, 487, 338, null);
       addKeyTime(12, 554, 323, null);
       addKeyTime(13, 500, 238, null);
       addKeyTime(14, 577, 206, null);
       addKeyTime(15, 632, 155, null);
       addKeyTime(16, 480, 151, null);
       addKeyTime(19, 301,  88, null);
       addKeyTime(21, 233, 149, null);
       addKeyTime(22, 147, 181, null);
       addKeyTime(30, 164, 210, null);
       setEndState(REMAIN);
    }

    private void addKeyTime(int time, int x, int y,
                            Content content)
    {
       addKeyTime(time*1000, new Point2D.Double((double)x, (double)y),
                  null, new Double(1.0), content);
    }
}
```



`BusOnRoute` – Demonstration

In `examples/chapter`:

BusOnRoute.html

```
java -cp BusOnRoute.jar BusOnRouteApplication
```

Tweening the Visual Content

- Sampled:

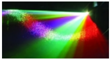
  Specify a raster for each key frame.

  Tweening from one to the next.

- Described:

  Specifying a shape (or shapes) for each key frame.

  Tweening from one to the next.

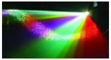


Tweening Sampled Static Content

- Use an object that has two component `statik.Content` objects.

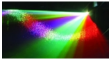
  `statik.CompositeContent` provides this capability but it a little easier to use a simple (i.e., non-hierarchical) collection.

- Combine them with alpha blending.



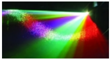
sampled.AggregateContent – Structure

```
package visual.statik.sampled;

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

public class      AggregateContent
       extends    visual.statik.AbstractAggregateContent<Content>
       implements TransformableContent
{
    public AggregateContent()
    {
       super();
    }
}
```



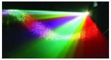
sampled.AggregateContent – setBufferedImageOp()

```
public void setBufferedImageOp(BufferedImageOp op)
{
   Iterator<Content>  i;

   i = iterator();
   while (i.hasNext())
   {
      i.next().setBufferedImageOp(op);
   }
}
```



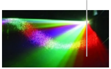
sampled.AggregateContent – setComposite()

```
   public void setComposite(Composite c)
   {
      Content     content;

      content = components.getLast();
      content.setComposite(c);
   }
```



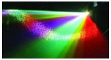
SampledSprite – Structure

```
package visual.dynamic.described;

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

import visual.statik.sampled.AggregateContent;
import visual.statik.sampled.Content;
import visual.statik.sampled.TransformableContent;

public class SampledSprite extends TweeningSprite
{
    private   AggregateContent   tweened;
    private   Vector<Content>    content;

    public SampledSprite()
    {
       super();

       content = new Vector<Content>();
    }

    public void addKeyTime(int keyTime, Point2D location,
                           Double rotation, Double scaling,
                           Content c)
    {
       int         index;
```



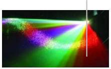
SampledSprite – Structure (cont.)

```
       index = super.addKeyTime(keyTime, location, rotation, scaling);

       if (index >= 0)
       {
          // If c is null then re-use the last Content
          if (c==null) c = content.get(index-1);

          content.insertElementAt(c, index);
       }
    }
}
```



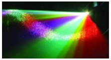
SampledSprite – getContent()

```
protected visual.statik.TransformableContent getContent()
{
   AggregateContent      aggregate;
   Content               currentContent, nextContent;
   float                 alpha;
   int                   current, next;
   visual.statik.TransformableContent   result;

   result  = null;
   current = getKeyTimeIndex();
   next    = getNextKeyTimeIndex();

   if (visible && (current >= 0))
   {
      currentContent = content.get(current);
      nextContent    = content.get(next);

      if ((nextContent != null) &&
          (currentContent != nextContent))
      {
         aggregate = new AggregateContent();
         aggregate.add(currentContent);
         aggregate.add(nextContent);

         // Setup alpha blending
         alpha = (float)getInterpolationFraction();

         aggregate.setComposite(
```



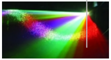
SampledSprite – getContent() (cont.)

```
                       AlphaComposite.getInstance(
                                AlphaComposite.SRC_OVER,
                                alpha));

            result = aggregate;
         }
         else
         {
            result = currentContent;
         }
      }
      return result;
   }
```



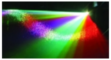
A Crystal Ball

```
import java.awt.geom.Point2D;

import io.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class CrystalBall extends SampledSprite
{
    public CrystalBall()
    {
        super();
        Content          content;
        ContentFactory   factory;
        ResourceFinder   finder;

        finder  = ResourceFinder.createInstance(this);
        factory = new ContentFactory(finder);


        content = factory.createContent("crystalball01.png");
        addKeyTime(  500,   0.0, 350.0, -0.75, content);
        addKeyTime( 4000, 100.0, 200.0, -0.30, null);

        content = factory.createContent("crystalball02.png");
        addKeyTime( 7500, 200.0,  50.0,  0.00, content);

        setEndState(REMAIN);
    }
```

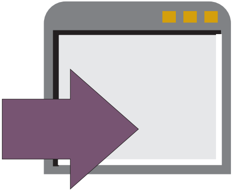


A Crystal Ball (cont.)

```
    private void addKeyTime(int time, double x, double y,
                             double r, Content content)
    {
       addKeyTime(time, new Point2D.Double(x, y), new Double(r),
                   new Double(1.0), content);
    }
}
```



A Crystal Ball – Demonstration

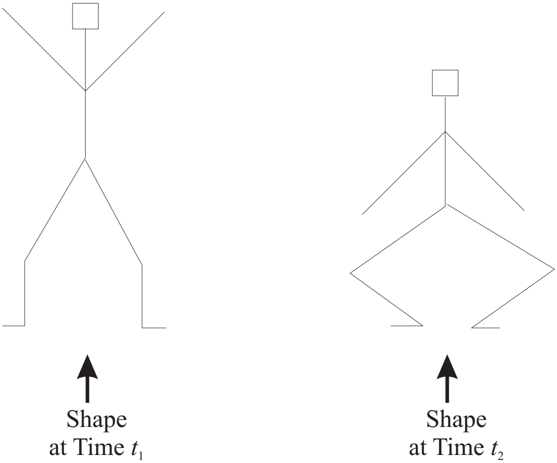


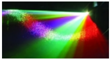
In `examples/chapter`:

CrystalBall.html

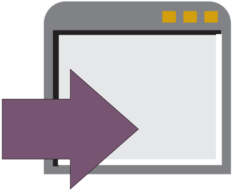
```
java -cp CrystalBall.jar CrystalBallApplication
```

Shape Tweening





Shape Tweening – Demonstration



In `examples/chapter`:

BuzzyJumping.html

```
java -cp BuzzyJumping.jar BuzzyJumpingApplication
```

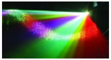
Shape Tweening (cont.)

- Most Common Approach:

  Tween the location of each of the points that defines the shape.

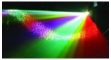
- A Helpful Participant:

  The `PathIterator` interface which provides access to "move to" and "draw to" segments.



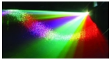
described.Content – PathIterator

```
public PathIterator getPathIterator(boolean transformed)
{
   if (transformed)
      return transformedShape.getPathIterator(IDENTITY);
   else
      return originalShape.getPathIterator(IDENTITY);
}
```



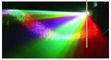
described.AggregateContent – Structure

```
package visual.statik.described;

import java.awt.*;
import java.util.Iterator;

public class      AggregateContent
       extends    visual.statik.AbstractAggregateContent<Content>
       implements TransformableContent
{
    public AggregateContent()
    {
       super();
    }
}
```



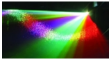
described.AggregateContent – Setters

```
   public void setColor(Color color)
   {
      Iterator<Content>  i;

      i = iterator();
      while (i.hasNext())
      {
         i.next().setColor(color);
      }
   }

   public void setPaint(Paint paint)
   {
      Iterator<Content>  i;

      i = iterator();
      while (i.hasNext())
      {
         i.next().setPaint(paint);
      }
   }

   public void setStroke(Stroke stroke)
   {
      Iterator<Content>  i;

      i = iterator();
      while (i.hasNext())
```



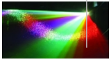
described.AggregateContent – Setters (cont.)

```
       {
          i.next().setStroke(stroke);
       }
    }
```



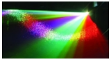
DescribedSprite – Structure

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;
import java.util.Iterator;
import java.util.Vector;

import visual.statik.described.*;

public class DescribedSprite extends TweeningSprite
{
    private   AggregateContent          tweened;
    private   Vector<AggregateContent>  content;

    public DescribedSprite()
    {
       content = new Vector<AggregateContent>();
       tweened = new AggregateContent();
    }

    public void addKeyTime(int keyTime, Point2D location,
                           Double rotation, Double scaling,
                           AggregateContent ctc)
    {
       int         index;

       index = super.addKeyTime(keyTime, location, rotation, scaling);
```



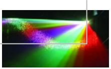
DescribedSprite – Structure (cont.)

```
       if (index >= 0)
       {
          // If ctc is null then re-use the last CompositeContent
          if (ctc == null) ctc = content.get(index-1);

          content.insertElementAt(ctc, index);
       }

    }
}
```



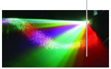
DescribedSprite – getContent()

```
    public visual.statik.TransformableContent getContent()
    {
       int                     current, next;
       AggregateContent        currentCTC, nextCTC, result;


       current = getKeyTimeIndex();
       next    = getNextKeyTimeIndex();

       result = null;

       if (current >= 0)
       {
          currentCTC = content.get(current);
          nextCTC    = content.get(next);

          result     = currentCTC;

          if (currentCTC != nextCTC)
          {
             tweenShape(currentCTC, nextCTC, getInterpolationFraction());
             result = tweened;
          }
       }

       return result;
    }
```



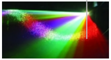
DescribedSprite – tweenShape()

```
protected void tweenShape(AggregateContent a,
                          AggregateContent b,
                          double frac)
{
   Color                       color;
   float[]                     coords, coordsA, coordsB;
   GeneralPath                 gp;
   int                         seg;
   Iterator<Content>           iterA, iterB;
   PathIterator                piA, piB;
   Paint                       paint;
   Content                     shapeA, shapeB;
   Stroke                      stroke;


   tweened = new AggregateContent();

   coordsA = new float[6];
   coordsB = new float[6];
   coords  = new float[6];

   iterA = a.iterator();
   iterB = b.iterator();

   // Loop over all of the TransformableContent objects
   // in the AggregateContent
   while (iterA.hasNext())
   {
```



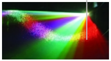
DescribedSprite – tweenShape() (cont.)

```
shapeA = iterA.next();
if (iterB.hasNext()) shapeB = iterB.next();
else                 shapeB = shapeA;

piA = shapeA.getPathIterator(false);
piB = shapeB.getPathIterator(false);

gp = new GeneralPath();
gp.setWindingRule(piA.getWindingRule());


// Loop over all of the segments in the
// TransformableContent object
while (!piA.isDone())
{
   seg = piA.currentSegment(coordsA);
   if (piB.isDone()) // Use the coordinates of the first shape
   {
      for (int i=0; i < coordsA.length; i++)
         coords[i] = coordsA[i];
   }
   else           // Interpolate the coordinates
   {
      piB.currentSegment(coordsB);

      for (int i=0; i < coordsA.length; i++)
      {
         coords[i] = coordsA[i] +
```



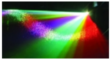
DescribedSprite – tweenShape() (cont.)

```
                  (float)frac*(coordsB[i] - coordsA[i]);
   }
}

// Add to the General Path object
if      (seg == PathIterator.SEG_MOVETO)
{
   gp.moveTo(coords[0], coords[1]);
}
else if (seg == PathIterator.SEG_LINETO)
{
   gp.lineTo(coords[0], coords[1]);
}
else if (seg == PathIterator.SEG_QUADTO)
{
   gp.quadTo(coords[0], coords[1], coords[2], coords[3]);
}
else if (seg == PathIterator.SEG_CUBICTO)
{
   gp.curveTo(coords[0], coords[1],
              coords[2], coords[3],
              coords[4], coords[5]);
}
else if (seg == PathIterator.SEG_CLOSE)
{
   gp.closePath();
}
```



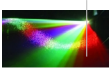
DescribedSprite – tweenShape() (cont.)

```
             piA.next();
             piB.next();
          }

          paint  = shapeA.getPaint();
          color  = shapeA.getColor(); // This could also be tweened
          stroke = shapeA.getStroke();

          tweened.add(new Content(gp, color, paint, stroke));
       }
    }
```



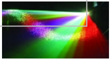
A JumboTron

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

import app.*;
import io.*;
import visual.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class   DynamicJumboTronApp
       extends AbstractMultimediaApp
{
    public void init()
    {
       BuzzyOnMars                       buzzy;
       Content                           mars;
       ContentFactory                    factory;
       JPanel                            contentPane;
       ResourceFinder                    finder;
       ScaledVisualizationRenderer       renderer2;
       Stage                             stage;
       VisualizationView                 view1, view2;

       finder  = ResourceFinder.createInstance(this);
       factory = new ContentFactory(finder);

       // The Stage for Buzzy
```



A JumboTron (cont.)

```
       stage = new Stage(50);
       stage.setBackground(Color.white);
       stage.setRestartTime(7000);
       view1 = stage.getView();
       view1.setBounds(0,0,640,480);

       renderer2 = new ScaledVisualizationRenderer(
                       new PlainVisualizationRenderer(), 640.0, 480.0);
       view2 = new VisualizationView(stage, renderer2);
       view2.setBounds(50,50,160,120);
       stage.addView(view2);

       mars = factory.createContent("mars.png");
       stage.add(mars);

       // Buzzy
       buzzy = new BuzzyOnMars();
       stage.add(buzzy);

       // The content pane
       contentPane = (JPanel)rootPaneContainer.getContentPane();
       contentPane.add(view2);
       contentPane.add(view1);

       stage.start();
    }
}
```

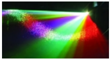


A JumboTron – Demonstration

In `examples/chapter`:

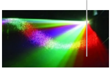
DynamicJumboTron.html

```
java -cp DynamicJumboTron.jar DynamicJumboTronApplication
```



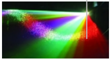
Picture-in-Picture

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

import app.*;
import io.*;
import visual.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class   DynamicPIPApp
       extends AbstractMultimediaApp
{
    public void init()
    {
       Airplane                      plane;
       BuzzyOnMars                   buzzy;
       Content                       mars;
       ContentFactory                factory;
       JPanel                        contentPane;
       ResourceFinder                finder;
       Stage                         stage1, stage2;
       VisualizationView             view1, view2;

       finder  = ResourceFinder.createInstance(this);
       factory = new ContentFactory(finder);

       // The Stage for Buzzy
```



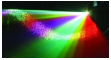
Picture-in-Picture (cont.)

```
stage1 = new Stage(50);
stage1.setBackground(Color.white);
stage1.setRestartTime(7000);
view1 = stage1.getView();
view1.setRenderer(new ScaledVisualizationRenderer(
                         view1.getRenderer(),
                         640.0, 480.0));
view1.setBounds(0,0,640,480);

mars = factory.createContent("mars.png");
stage1.add(mars);

// Buzzy
buzzy = new BuzzyOnMars();
stage1.add(buzzy);

// The stage for the airplane
stage2 = new Stage(50);
view2 = stage2.getView();
view2.setRenderer(new ScaledVisualizationRenderer(
                         view2.getRenderer(),
                         640.0, 480.0));
view2.setBounds(50,50,160,120);
view2.setSize(160,120);
view2.setBackground(Color.white);
stage2.setRestartTime(12000);

// The Airplane
```

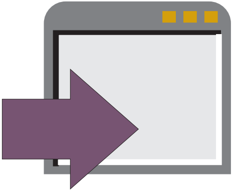


Picture-in-Picture (cont.)

```
        plane = new Airplane();
        stage2.add(plane);

        // The content pane
        contentPane = (JPanel)rootPaneContainer.getContentPane();
        contentPane.add(view2);
        contentPane.add(view1);

        stage1.start();
        stage2.start();
    }
}
```



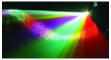
Picture-in-Picture – Demonstration



In `examples/chapter`:

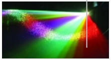
DynamicPIP.html

```
java -cp DynamicPIP.jar DynamicPIPApplication
```



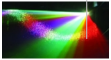
A Diptych

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

import app.*;
import io.*;
import visual.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class   DynamicDiptychApp
       extends AbstractMultimediaApp
{
    public void init()
    {
       BuzzyOnMars                    buzzy;
       Content                        mars;
       ContentFactory                 factory;
       JFrame                         window2;
       JPanel                         contentPane;
       ResourceFinder                 finder;
       Stage                          stage;
       VisualizationRenderer          renderer1, renderer2;
       VisualizationView              view1, view2;


       // The Stage for Buzzy
       stage = new Stage(50);
```



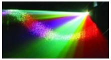
A Diptych (cont.)

```
stage.setBackground(Color.white);
stage.setRestartTime(7000);
view1 = stage.getView();
view1.setRenderer(new PartialVisualizationRenderer(
                         view1.getRenderer(),
                         0.0, 0.0));
view1.setBounds(0,0,320,480);

renderer2 = new PartialVisualizationRenderer(
                new PlainVisualizationRenderer(), 320.0, 0.0);
view2 = new VisualizationView(stage, renderer2);
view2.setBounds(0,0,320,480);
stage.addView(view2);

finder = ResourceFinder.createInstance(this);
factory = new ContentFactory(finder);

mars = factory.createContent("mars.png");
stage.add(mars);

// Buzzy
buzzy = new BuzzyOnMars();
stage.add(buzzy);

// The content pane for the main window
contentPane = (JPanel)rootPaneContainer.getContentPane();
contentPane.add(view1);
```

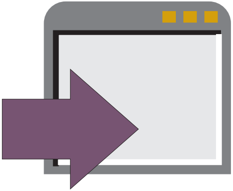


A Diptych (cont.)

```
        // The content pane for the other window
        window2 = new JFrame();
        window2.setSize(320,480);
        window2.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        contentPane = (JPanel)window2.getContentPane();
        contentPane.add(view2);
        window2.setVisible(true);

        stage.start();
    }
}
```



A Diptych – Demonstration



In `examples/chapter`:

```
java -cp DynamicDiptych.jar DynamicDiptychApplication
```

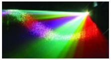
Adding Special Effects to Sampled Dynamic Visual Content

- The Objective:

  Add “sprites” to a “movie”.

- What’s Needed?:

  What’s Needed?



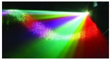
Adding Special Effects to Sampled Dynamic Visual Content

- The Objective:

  Add “sprites” to a “movie”.

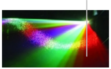
- What’s Needed?:

  The `Screen` object’s `Visualization` and the `Stage` need to render to the same `VisualizationView`.



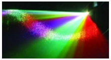
SpecialEffectsRenderer

```
package visual.dynamic;

import java.awt.*;
import java.util.*;
import javax.swing.*;

import visual.*;
import visual.statik.*;
import visual.dynamic.described.Sprite;

public class      SpecialEffectsRenderer
       implements VisualizationRenderer
{
    private Visualization           stage;
    private VisualizationRenderer   decorated;

    public SpecialEffectsRenderer(VisualizationRenderer decorated,
                                  Visualization         stage)
    {
       this.decorated = decorated;
       this.stage     = stage;
    }

    public void postRendering(Graphics          g,
                              Visualization     model,
                              VisualizationView view)
    {
       decorated.postRendering(g, model, view);
```



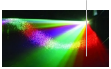
SpecialEffectsRenderer (cont.)

```
    }

    public void preRendering(Graphics          g,
                             Visualization     model,
                             VisualizationView view)
    {
       decorated.preRendering(g, model, view);
    }

    public void render(Graphics          g,
                       Visualization     model,
                       VisualizationView view)
    {
       decorated.render(g, model, view);
       decorated.render(g, stage, view);
    }
}
```



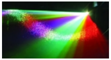
SpecialEffectsScreen

```
package visual.dynamic;

import event.*;
import visual.*;
import visual.dynamic.described.*;
import visual.dynamic.sampled.*;

public class SpecialEffectsScreen extends Screen
{
    SpecialEffectsRenderer    renderer;
    Visualization             stage;

    public SpecialEffectsScreen()
    {
       super();
       stage.setView(getView());
    }

    public void add(Sprite sprite)
    {
       // Make the Sprite a MetronomeListener
       metronome.addListener(sprite);

       // Treat the Sprite as a SimpleContent and
       // add it to the Visualization
       stage.add(sprite);
    }
```



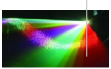
SpecialEffectsScreen (cont.)

```
    protected VisualizationView createDefaultView()
    {
       stage = new Visualization();

       renderer = new SpecialEffectsRenderer(
                      new ScreenRenderer(
                          new PlainVisualizationRenderer()),
                      stage);

       return new VisualizationView(this, renderer);
    }
}
```



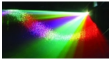
A Special Effect - A Bee

```
import java.awt.geom.*;
import java.awt.image.*;

import io.*;
import visual.dynamic.described.*;
import visual.dynamic.sampled.*;
import visual.statik.sampled.*;

public class   Bee
       extends SampledSprite
{
    public Bee()
    {
        super();
        Content            content;
        ContentFactory     factory;
        ResourceFinder     finder;

        finder  = ResourceFinder.createInstance(this);
        factory = new ContentFactory(finder);
        content = factory.createContent("bee.png", 4);
        addKeyFrame(    1, 173.0, 118.0,  0.00, 0.20, content);
        addKeyFrame(   45, 166.0, 120.0,  0.00, 0.35, null);
        addKeyFrame(  100, 148.0, 105.0,  0.00, 0.50, null);
        addKeyFrame(  115, 230.0,  90.0,  0.00, 0.75, null);
        addKeyFrame(  150, 245.0, 143.0,  0.00, 1.00, null);

        setEndState(REMOVE);
```



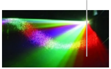
A Special Effect - A Bee (cont.)

```
    }

    private void addKeyFrame(int frame, double x, double y,
                             double r, double s, Content c)
    {
       int     time;

       time = frame * Screen.DEFAULT_FRAME_DELAY;

       addKeyTime(time, new Point2D.Double(x, y), new Double(r),
                  new Double(s), c);
    }
}
```



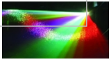
An Example

```
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;

import app.*;
import visual.*;
import io.ResourceFinder;
import visual.dynamic.*;
import visual.dynamic.described.*;
import visual.dynamic.sampled.*;
import visual.statik.*;
import visual.statik.described.*;
import visual.statik.sampled.*;

public class   SpecialEffectsApp
       extends AbstractMultimediaApp
{
    public void init()
    {
       Bee                                   bee;
       visual.statik.sampled.Content         content;
       ContentFactory                        factory;
       JPanel                                contentPane;
       ResourceFinder                        finder;
       SpecialEffectsScreen                  screen;
       SimpleContent[]                       frames;
       String[]                              names;
       VisualizationView                     view;
```

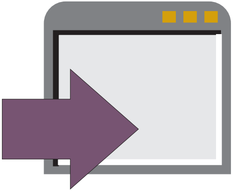


An Example (cont.)

```
       screen = new SpecialEffectsScreen();
       screen.setRepeating(true);

       view = screen.getView();
       view.setBounds(0,0,320,240);

       contentPane = (JPanel)rootPaneContainer.getContentPane();
       contentPane.add(view);

       finder = ResourceFinder.createInstance(this);

       names = finder.loadResourceNames("scribble.txt");
       factory = new ContentFactory(finder);
       frames = factory.createContents(names, 4);

       for (int i=0; i<frames.length; i++)
       {
          screen.add(frames[i]);
       }

       bee = new Bee();
       screen.add(bee);

       screen.start();
    }
}
```



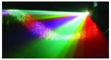
Special Effects – Demonstration



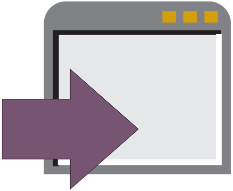
In `examples/chapter`:

SpecialEffects.html

```
java -cp SpecialEffects.jar SpecialEffectsApplication -Xmx256m
```



Putting it All Together – Demonstration



In `examples/chapter`:

SpecialEffectsPIP.html

```
java -cp SpecialEffectsPIP.jar SpecialEffectsPIPApplication -Xmx256m
```