Sprites & Transforms

Computer Game Development

David Cairns

Sprites

If we combine an animation with independent movement into a class, we get a Sprite.

A sprite can move around the screen independently and will update its own position and animation based on its speed and time elapsed between updates.

Properties

- Animation: A set of animation frames and durations
- Position: X & Y co-ordinates on screen
- Velocity
  - A speed and direction OR
  - Horizontal & vertical speed (in pixels/millisecond)

Sprite Class - 1

```
public class Sprite {

    // The current Animation to use for this sprite
    private Animation anim;

    // Position (pixels)
    private float x;
    private float y;

    // Velocity (pixels per millisecond)
    private float dx;
    private float dy;

    // Dimensions of the sprite
    private float height;
    private float width;
    private float radius;
```

Sprite Class - 2

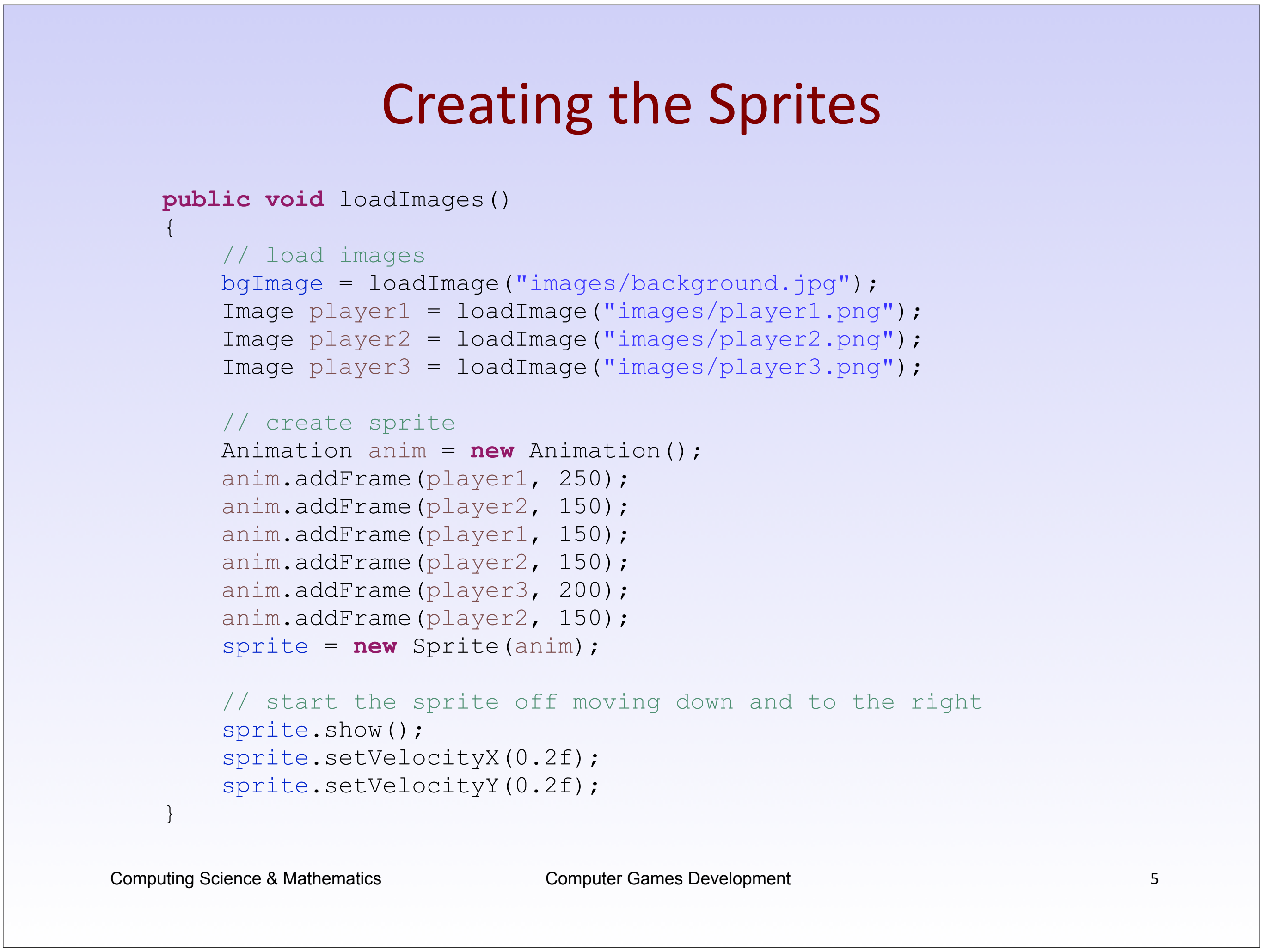
```
  // Create a new Sprite object with the specified Animation.
  public Sprite(Animation anim)
  {
       this.anim = anim;
       render = false;
       scale = 1.0f;
       rotation = 0.0f;
  }

   // Update the Sprite's Animation and position based on elapsedTime.
  public void update(long elapsedTime)
   {
     if (!render) return;

     x += dx * elapsedTime;
     y += dy * elapsedTime;
     anim.update(elapsedTime);

  }

 … Lots of get and set methods for the sprite properties
```

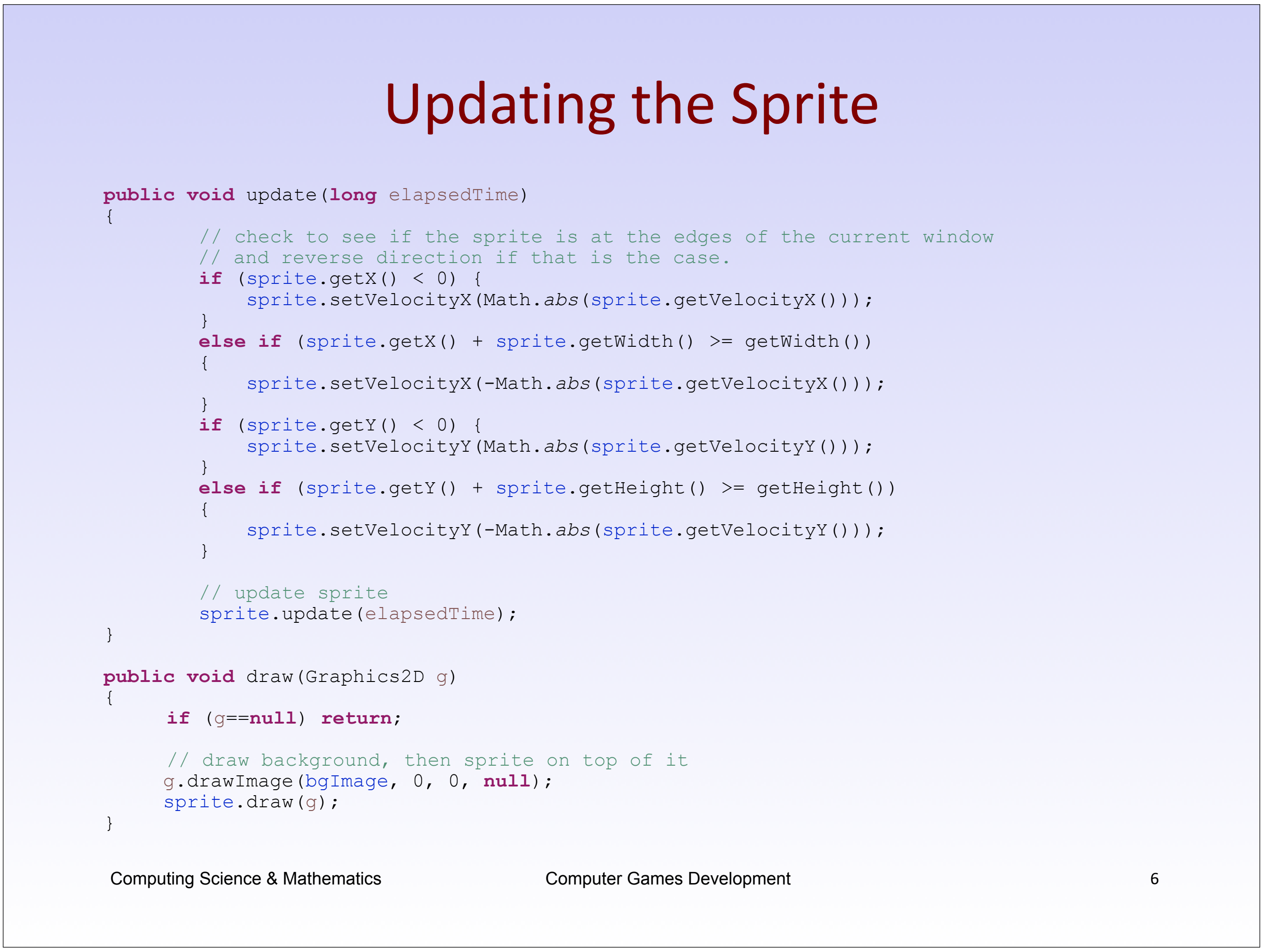


Creating the Sprites

```
public void loadImages()
{
    // load images
    bgImage = loadImage("images/background.jpg");
    Image player1 = loadImage("images/player1.png");
    Image player2 = loadImage("images/player2.png");
    Image player3 = loadImage("images/player3.png");

    // create sprite
    Animation anim = new Animation();
    anim.addFrame(player1, 250);
    anim.addFrame(player2, 150);
    anim.addFrame(player1, 150);
    anim.addFrame(player2, 150);
    anim.addFrame(player3, 200);
    anim.addFrame(player2, 150);
    sprite = new Sprite(anim);

    // start the sprite off moving down and to the right
    sprite.show();
    sprite.setVelocityX(0.2f);
    sprite.setVelocityY(0.2f);
}
```



Updating the Sprite

```
public void update(long elapsedTime)
{
        // check to see if the sprite is at the edges of the current window
        // and reverse direction if that is the case.
        if (sprite.getX() < 0) {
            sprite.setVelocityX(Math.abs(sprite.getVelocityX()));
        }
        else if (sprite.getX() + sprite.getWidth() >= getWidth())
        {
            sprite.setVelocityX(-Math.abs(sprite.getVelocityX()));
        }
        if (sprite.getY() < 0) {
            sprite.setVelocityY(Math.abs(sprite.getVelocityY()));
        }
        else if (sprite.getY() + sprite.getHeight() >= getHeight())
        {
            sprite.setVelocityY(-Math.abs(sprite.getVelocityY()));
        }

        // update sprite
        sprite.update(elapsedTime);
}

public void draw(Graphics2D g)
{
      if (g==null) return;

      // draw background, then sprite on top of it
      g.drawImage(bgImage, 0, 0, null);
      sprite.draw(g);
}
```

Sprite Code Example

Demo

- Single Sprite
- Multiple Sprites

Image Transforms

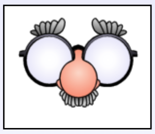
Consider operations on an image - we may wish to

- Translate
  - Move image within its frame
- Flip
  - Flip about an axis, e.g. horizontal or vertical
- Rotate
  - Spin around a point
- Scale
  - Magnify or shrink to a different size
- Shear

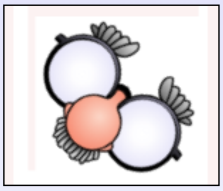
These are Affine Transformations : They preserve the "straightness" and "parallel" features of lines within the image.

Flip

Scale



Rotate



Shear

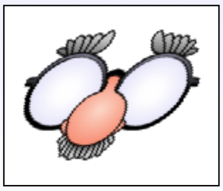


Image Transforms in Java

Transforms are useful when rotating a sprite around or making a smaller, iconic version of an image. You could do this ‘offline’ with a graphics package but it would take up a lot of memory (and be tedious).

Java uses the AffineTransform class to apply these types of transform to an image when it is rendered.

```
AffineTransform transform = new AffineTransform();
transform.scale(2,2);         // Make image twice as big in X & Y dimension
transform.translate(100,50);  // Move the image 100 to the right, 50 down
g.drawImage(image,transform,null); // Apply transform to the image and draw it
```

- Transforms must be applied in reverse order!
- Graphics2D provides the transform image drawing methods

The AffineTransform Class

The available transform operations for the AffineTransform class are:

- **translate**(double x, double y)
  - Move image across x pixels and down y pixels
- **scale**(double x, double y)
  - Increase / decrease dimensions such that new x dimension = old x dimension * x and new y dimension = old y dimesion * y
  - Note that if you use scale with negative values, this effectively **flips** the image (e.g. scale(-1,1) flips image about the x axis)
- **rotate**(double angle)
  - Rotate the image by the given angle in radians (1 radian = 180/π°)
- **shear**(double x, double y)
  - Shift the image in the x and y axis relative to their proportionate values

Implementation of Transforms

An AffineTransform is implemented as a 3x3 matrix of values or functions that are applied to a given x,y co-ordinate:

```
[ x']   [ m00 m01 m02 ] [ x ]   [ m00*x + m01*y + m02*1 ]
[ y'] = [ m10 m11 m12 ] [ y ] = [ m10*x + m11*y + m12*1 ]
[ 1 ]   [ 0   0   1   ] [ 1 ]   [   0*x +   0*y +   1*1 ]
```

They effectively define how much each original x and y point should be adjusted to implement the required transform.

Translate

Take the simplest... Translate, defined as:

```
[ 1 0 tx ]
[ 0 1 ty ]
[ 0 0 1  ]
```

```
[ x']    [ 1   0   tx ] [ x ]    [ 1*x + 0*y + tx*1 ]
[ y'] =  [ 0   1   ty ] [ y ] =  [ 0*x + 1*y + ty*1 ]
[ 1 ]    [ 0   0    1 ] [ 1 ]    [ 0*x + 0*y + 1*1  ]
```

```
[ x']    [ x + tx ]
[ y'] =  [ y + ty ]
[ 1 ]    [   1    ]
```

Scale

Now change slightly... Scale, defined as:

```
[ sx 0  0 ]
[ 0  sy 0 ]
[ 0   0 1 ]
```

```
[ x’]   [ sx 0   0 ] [ x ]        [ sx*x + 0*y + 0*1 ]
[ y'] = [ 0  sy  0 ] [ y ] =      [ 0*x + sy*y + 0*1 ]
[ 1 ]   [ 0  0   1 ] [ 1 ]        [ 0*x + 0*y  + 1*1 ]
```

```
[ x’]   [ sx * x ]
[ y'] = [ sy * y ]
[ 1 ]   [   1    ]
```

See the AffineTransform entry in the Java API guide for the matrices used in the other transforms

Translate & Scale

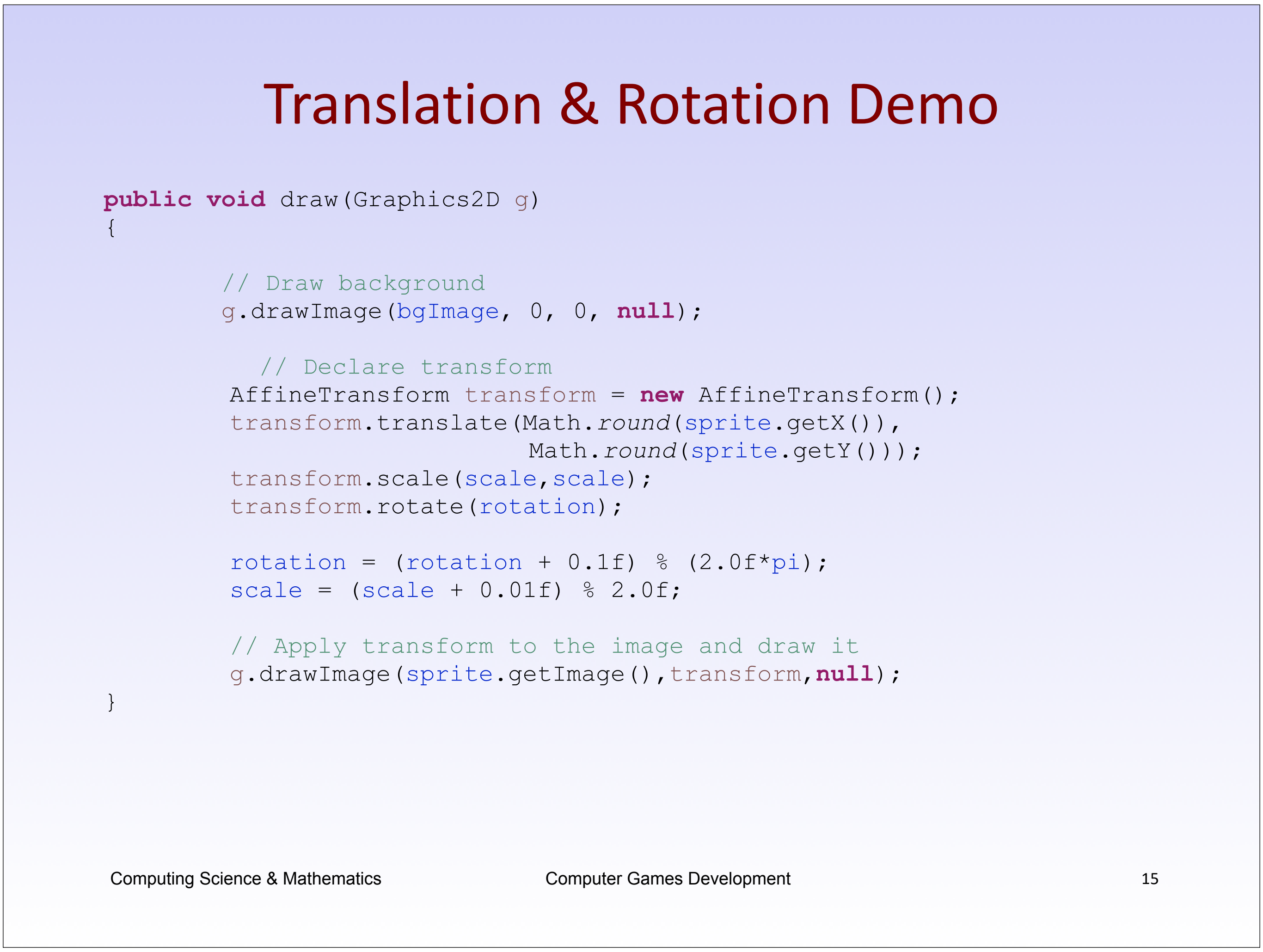
Now change slightly... Scale, defined as:

```
[ sx 0  tx ]
[ 0  sy ty ]
[ 0   0 1  ]
```

```
[ x’]   [ sx 0   tx ] [ x ]       [ sx*x + 0*y + tx*1 ]
[ y'] = [ 0  sy  ty ] [ y ] =     [ 0*x + sy*y + ty*1 ]
[ 1 ]   [ 0  0   1  ] [ 1 ]       [ 0*x + 0*y  + 1*1 ]
```

```
[ x’]   [ sx * x   + tx]
[ y'] = [ sy * y   + ty]
[ 1 ]   [      1       ]
```

See the AffineTransform entry in the Java API guide for the matrices used in the other transforms



Translation & Rotation Demo

```
public void draw(Graphics2D g)
{

        // Draw background
        g.drawImage(bgImage, 0, 0, null);

           // Declare transform
         AffineTransform transform = new AffineTransform();
         transform.translate(Math.round(sprite.getX()),
                             Math.round(sprite.getY()));
         transform.scale(scale,scale);
         transform.rotate(rotation);

         rotation = (rotation + 0.1f) % (2.0f*pi);
         scale = (scale + 0.01f) % 2.0f;

         // Apply transform to the image and draw it
         g.drawImage(sprite.getImage(),transform,null);
}
```