



```
do {
    Graphics g = null;
    try {
        g = bs.getDrawGraphics();
        g.clearRect( 0, 0, getWidth(), getHeight() );
        render( g );
    } finally {
        if( g != null ) {
            g.dispose();
        }
    }
} while( bs.contentsRestored() );
bs.show();
} while( bs.contentsLost() );
}
private void sleep( long sleep ) {
try {
    Thread.sleep( sleep );
} catch( InterruptedException ex ) { }
}
private void processInput() {
    keyboard.poll();
    mouse.poll();
    if( keyboard_KeyDownOnce( KeyEvent.VK_SPACE ) ) {
        System.out.println("VK_SPACE");
    }
    // if button is pressed for first time,
    // start drawing lines
    if( mouse.buttonDownOnce( MouseEvent.BUTTON1 ) ) {
        drawingLine = true;
    }
    // if the button is down, add line point
    if( mouse.buttonDown( MouseEvent.BUTTON1 ) ) {
        lines.add( mouse.getPosition() );
        // if the button is not down but we were drawing,
        // add a null to break up the lines
    } else if( drawingLine ) {
        lines.add( null );
        drawingLine = false;
    }
    // if 'C' is down, clear the lines
    if( keyboard_KeyDownOnce( KeyEvent.VK_C ) ) {
        lines.clear();
    }
}
```



```
private void render( Graphics g ) {
    colorIndex += mouse.getNotches();
    Color color = COLORS[ Math.abs( colorIndex % COLORS.length ) ];
    g.setColor( color );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 30, 30 );
    g.drawString( "Use mouse to draw lines", 30, 45 );
    g.drawString( "Press C to clear lines", 30, 60 );
    g.drawString( "Mouse Wheel cycles colors", 30, 75 );
    g.drawString( mouse.getPosition().toString(), 30, 90 );
    for( int i = 0; i < lines.size() - 1; ++i ) {
        Point p1 = lines.get( i );
        Point p2 = lines.get( i + 1 );
        // Adding a null into the list is used
        // for breaking up the lines when
        // there are two or more lines
        // that are not connected
        if( !( p1 == null || p2 == null ) )
            g.drawLine( p1.x, p1.y, p2.x, p2.y );
    }
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final SimpleMouseExample app = new SimpleMouseExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
```



## 2.4 相对鼠标移动

对于图 2.3 所示的当前的鼠标输入类来说，有一个问题。首先，它看上去似乎挺明显，但是，只有在鼠标位于窗口之中的时候，程序才接受鼠标事件。一旦鼠标离开了窗口，鼠标指针的坐标位置在应用程序中就变得不再有效。更为糟糕的是，在全屏模式中，当鼠标到达屏幕边缘的时候，它会直接停下来，而不会继续注册事件。根据应用程序的需要，可能需要相对鼠标移动。好在，更新鼠标输入类以支持相对鼠标移动并不难。

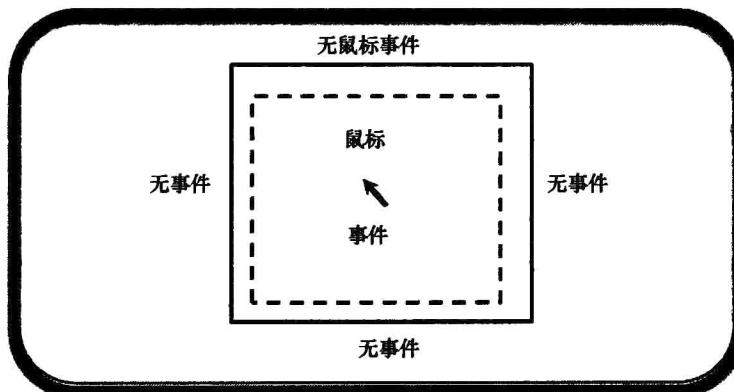


图 2.3 仅用于窗口的鼠标事件

RelativeMouseInput 类位于 javagames.util 包中，构建于前面示例中的类的基础之上，并且添加了相对鼠标移动。为了实现这一点，用 Robot 类来将鼠标保持在窗口的中央。还有监听鼠标事件的 Swing 组件，可以计算窗口的中央位置，并且从相对窗口坐标转换为绝对屏幕坐标。如果鼠标光标总是位于窗口的中央，那么，它可能不会离开，并且窗口将总是接受鼠标事件。如下代码保持鼠标居中：

```
// RelativeMouseInput.java
private Point getComponentCenter() {
    int w = component.getWidth();
    int h = component.getHeight();
    return new Point( w / 2, h / 2 );
}
private void centerMouse() {
    if( robot != null && component.isShowing() ) {
```



```
    Point center = getComponentCenter();
    SwingUtilities.convertPointToScreen( center, component );
    robot.mouseMove( center.x, center.y );
}
}
```

只有在运行时才会计算窗口的中央位置，因此即使窗口改变了大小，鼠标将仍然位于中央。相对中央位置必须转换为绝对屏幕坐标。不管窗口位于桌面上的何处，窗口左上角的像素都是(0,0)。如果在桌面上移动窗口会改变左上角像素的值，那么，图形化编程将会变得非常困难。尽管使用相对像素值会使得绘制较为容易，但把鼠标定位到窗口的中央并没有考虑窗口的位置，也没有考虑到将鼠标光标放置到距离窗口很远时，它会停止接受鼠标事件。使用 `SwingUtilities` 类转换得到屏幕坐标，从而解决这一问题。

要意识到将鼠标重新居中很重要，因为当鼠标的新位置和当前位置相同时，要求 `Robot` 类重新把鼠标定位到相同的位置，而这并不会产生新的鼠标事件。如果这种行为有变化的话，相对鼠标类将总是产生鼠标事件，即便鼠标并没有移动。即便鼠标行为将来不发生变化，在请求重新居中之前，也应先检查当前位置和新的位置是否不同，这种做法将会解决这个问题。

给 `RelativeMouseInput` 类添加一个标志，以允许相对和绝对鼠标移动：一款游戏在某些时候可能既需要绝对鼠标模式，也需要相对鼠标模式，因此，这个类允许在运行时切换。`mouseMoved()`方法添加了新的代码。如果是在相对模式中，距离将计算为与中心点之间的差距，然后让鼠标光标重新居中。由于鼠标坐标和组件坐标都是相对值，因此不需要转换这些值。最后，在轮询方法的过程中，鼠标位置可以是相对的或绝对的。在轮询方法中，`delta` 变量和所有其他变量一起重新设置。

```
package javagames.util;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class RelativeMouseInput
implements MouseListener, MouseMotionListener, MouseWheelListener {
    private static final int BUTTON_COUNT = 3;
    private Point mousePos;
    private Point currentPos;
    private boolean[] mouse;
    private int[] polled;
    private int notches;
    private int polledNotches;
    private int dx, dy;
```



```
private Robot robot;
private Component component;
private boolean relative;
public RelativeMouseInput( Component component ) {
    this.component = component;
    try {
        robot = new Robot();
    } catch( Exception e ) {
        // Handle exception [game specific]
        e.printStackTrace();
    }
    mousePos = new Point( 0, 0 );
    currentPos = new Point( 0, 0 );
    mouse = new boolean[ BUTTON_COUNT ];
    polled = new int[ BUTTON_COUNT ];
}
public synchronized void poll() {
    if( isRelative() ) {
        mousePos = new Point( dx, dy );
    } else {
        mousePos = new Point( currentPos );
    }
    dx = dy = 0;
    polledNotches = notches;
    notches = 0;
    for( int i = 0; i < mouse.length; ++i ) {
        if( mouse[i] ) {
            polled[i]++;
        } else {
            polled[i] = 0;
        }
    }
}
public boolean isRelative() {
    return relative;
}
public void setRelative( boolean relative ) {
    this.relative = relative;
    if( relative ) {
        centerMouse();
    }
}
```



```
public Point getPosition() {
    return mousePos;
}

public int getNotches() {
    return polledNotches;
}

public boolean buttonDown( int button ) {
    return polled[ button - 1 ] > 0;
}

public boolean buttonDownOnce( int button ) {
    return polled[ button - 1 ] == 1;
}

public synchronized void mousePressed( MouseEvent e ) {
    int button = e.getButton() - 1;
    if( button >= 0 && button < mouse.length ) {
        mouse[ button ] = true;
    }
}

public synchronized void mouseReleased( MouseEvent e ) {
    int button = e.getButton() - 1;
    if( button >= 0 && button < mouse.length ) {
        mouse[ button ] = false;
    }
}

public void mouseClicked( MouseEvent e ) {
    // Not needed
}

public synchronized void mouseEntered( MouseEvent e ) {
    mouseMoved( e );
}

public synchronized void mouseExited( MouseEvent e ) {
    mouseMoved( e );
}

public synchronized void mouseDragged( MouseEvent e ) {
    mouseMoved( e );
}

public synchronized void mouseMoved( MouseEvent e ) {
    if( isRelative() ) {
        Point p = e.getPoint();
        Point center = getComponentCenter();
        dx += p.x - center.x;
    }
}
```



```
        dy += p.y - center.y;
        centerMouse();
    } else {
        currentPos = e.getPoint();
    }
}

public synchronized void mouseWheelMoved( MouseWheelEvent e ) {
    notches += e.getWheelRotation();
}

private Point getComponentCenter() {
    int w = component.getWidth();
    int h = component.getHeight();
    return new Point( w / 2, h / 2 );
}

private void centerMouse() {
    if( robot != null && component.isShowing() ) {
        Point center = getComponentCenter();
        SwingUtilities.convertPointToScreen( center, component );
        robot.mouseMove( center.x, center.y );
    }
}
}
```

RelativeMouseExample 位于 javagames.input 包中，如图 2.4 所示，它针对鼠标输入类测试更新的新功能。

如下的代码，通过将光标图像设置为在运行时创建的一个空的光标，从而关闭鼠标光标。

```
// RelativeMouseExample .java
private void disableCursor() {
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image image = tk.createImage( "" );
    Point point = new Point( 0, 0 );
    String name = "CanBeAnything";
    Cursor cursor = tk.createCustomCursor( image, point, name );
    setCursor( cursor );
}
```

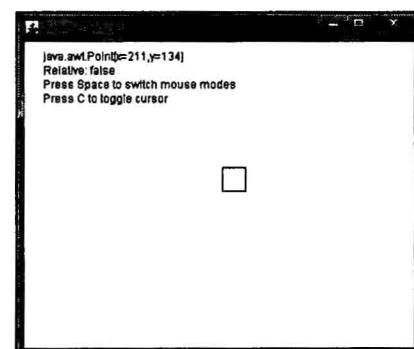


图 2.4 相对鼠标示例

这个示例的 render()方法显示了帮助文本，计算了帧速率，并且在屏幕上绘制了矩形。在 processInput()方法内部，空格将鼠标模式从绝对模式切换为相对模式。C 键用来显示或



隐藏鼠标光标。在当前的鼠标位置用来更新方块位置的时候，相对值添加到了该位置，而绝对值替代了之前的值。这段代码确保了方框不会离开屏幕，如果它在任何一个方向走得太远的话，都会折返其位置。

```
package javagames.input;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javagames.util.*;
import javax.swing.*;

public class RelativeMouseExample extends JFrame
    implements Runnable {
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private Canvas canvas;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private Point point = new Point( 0, 0 );
    private boolean disableCursor = false;
    public RelativeMouseExample() {
        frameRate = new FrameRate();
    }
    protected void createAndShowGUI() {
        canvas = new Canvas();
        canvas.setSize( 640, 480 );
        canvas.setBackground( Color.BLACK );
        canvas.setIgnoreRepaint( true );
        getContentPane().add( canvas );
        setTitle( "Relative Mouse Example" );
        setIgnoreRepaint( true );
        pack();
        // Add key listeners
        keyboard = new KeyboardInput();
        canvas.addKeyListener( keyboard );
        // Add mouse listeners
        // For full screen : mouse = new RelativeMouseInput( this );
        mouse = new RelativeMouseInput( canvas );
        canvas.addMouseListener( mouse );
        canvas.addMouseMotionListener( mouse );
        canvas.addMouseWheelListener( mouse );
        setVisible( true );
```



## 2.4 相对鼠



```
    canvas.createBufferStrategy( 2 );
    bs = canvas.getBufferStrategy();
    canvas.requestFocus();
    gameThread = new Thread( this );
    gameThread.start();
}
public void run() {
    running = true;
    frameRate.initialize();
    while( running ) {
        gameLoop();
    }
}
private void gameLoop() {
    processInput();
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
            try {
                g = bs.getDrawGraphics();
                g.clearRect( 0, 0, getWidth(), getHeight() );
                render( g );
            } finally {
                if( g != null ) {
                    g.dispose();
                }
            }
        }
        while( bs.contentsRestored() );
        bs.show();
    } while( bs.contentsLost() );
}
private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}
private void processInput() {
    keyboard.poll();
    mouse.poll();
    Point p = mouse.getPosition();
```



```
if( mouse.isRelative() ) {
    point.x += p.x;
    point.y += p.y;
} else {
    point.x = p.x;
    point.y = p.y;
}
// Wrap rectangle around the screen
if( point.x + 25 < 0 )
    point.x = canvas.getWidth() - 1;
else if( point.x > canvas.getWidth() - 1 )
    point.x = -25;
if( point.y + 25 < 0 )
    point.y = canvas.getHeight() - 1;
else if( point.y > canvas.getHeight() - 1 )
    point.y = -25;
// Toggle relative
if( keyboard_KeyDownOnce( KeyEvent.VK_SPACE ) ) {
    mouse.setRelative( !mouse.isRelative() );
}
// Toggle cursor
if( keyboard_KeyDownOnce( KeyEvent.VK_C ) ) {
    disableCursor = !disableCursor;
    if( disableCursor ) {
        disableCursor();
    } else {
        // setCoursor( Cursor.DEFAULT_CURSOR ) is deprecated
        setCursor( new Cursor( Cursor.DEFAULT_CURSOR ) );
    }
}
private void render( Graphics g ) {
    g.setColor( Color.GREEN );
    frameRate.calculate();
    g.drawString( mouse.getPosition().toString(), 20, 20 );
    g.drawString( "Relative: " + mouse.isRelative(), 20, 35 );
    g.drawString( "Press Space to switch mouse modes", 20, 50 );
    g.drawString( "Press C to toggle cursor", 20, 65 );
    g.setColor( Color.WHITE );
    g.drawRect( point.x, point.y, 25, 25 );
}
private void disableCursor() {
    Toolkit tk = Toolkit.getDefaultToolkit();
    Image image = tk.createImage( "" );
```



```
Point point = new Point( 0, 0 );
String name = "CanBeAnything";
Cursor cursor = tk.createCustomCursor( image, point, name );
setCursor( cursor );
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final RelativeMouseExample app = new RelativeMouseExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
}
```

## 2.5 资源和延伸阅读

“How to Write a Key Listener,” 1995, <http://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html>.

“How to Write a Mouse Listener,” 1995, <http://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html>.

“How to Write a Mouse-Motion Listener,” 1995, <http://docs.oracle.com/javase/tutorial/uiswing/events/mousemotionlistener.html>.

# 第3章

## 变换

在 2D 图形中，可以对形状进行 4 种不同的变换，如图 3.1 所示。

- 平移——向上、向下、向左或向右移动。
- 旋转——将物体围绕着一个点旋转。
- 缩放——将物体放大或缩小。
- 切变——沿着一个或两个方向移动物体。

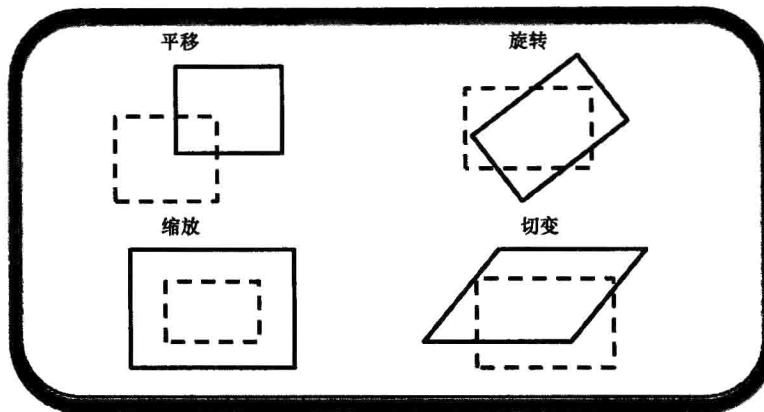


图 3.1 平移、旋转、缩放、切变

这些操作中的每一个，都可以以不同的方式进行。对于物体中的每一个坐标，可以手动完成操作，或者将操作组合成一个矩阵，然后一次性地应用。每种方法都有自己的优点和缺点，但是，要学习如何应用这些变换，动手计算每个坐标是最好的方法。

### 3.1 使用 Vector2f 类

Vector2f 类位于 javagames.util 包中，用于展示空间中的一个位置以及一个方向。注意，



这个 2D 坐标类有 3 个值：x、y 和 w。要使用矩阵变换来执行变换，还需要一个额外的坐标值，以便让矩阵数学能够发挥作用。因此，注意幕后那个额外的“w”坐标，我们将稍后介绍它。



熟悉向量算术的读者看到如下源代码的时候，可能会惊呼道：“等等！点和向量是不同的。点有一个  $w = 1$ ，而向量不能平移，并且不应该有一个  $w = 0$ ”。我后面将会介绍这些，所以请别担心。

平移非常简单。为了将坐标向上、向下、向左或向右移动，要给坐标加上移动的距离：

```
x = x + tx;  
y = y + ty;
```

缩放也很简单。用坐标值乘以一个缩放因子，就可以放大或缩小它们。

```
x = x * sx;  
y = y * sy;
```

切变稍微有点复杂，它将值相对于其他的坐标值移动。注意，应使用临时值来保存 x 的旧值，直到用它切变 y 值。如果没有这个临时值的话，将会使用更新后的值而不是最初的值来切变 y 值。要切变值，应给 x 值加上缩放后的 y 值，给 y 值加上缩放后的 x 值。

```
float tmp = x + sx * y;  
y = y + sy * x;  
x = tmp;
```

旋转更难理解一些。就像切变的示例，旋转也使用一个临时值，它像下面这样执行：

```
float tmp = x * cos(angle) - y * sin(angle);  
y = x * sin(angle) + y * cos(angle);  
x = tmp;
```

要理解这个公式的来源，需要理解极坐标，本章稍后将会介绍极坐标。现在，只需要假设它是有效的。

```
package javagames.util;  
  
public class Vector2f {  
    public float x;  
    public float y;  
    public float w;  
    public Vector2f() {  
        this.x = 0.0f;  
        this.y = 0.0f;  
        this.w = 1.0f; //!?
```



```
    }
    public Vector2f( Vector2f v ) {
        this.x = v.x;
        this.y = v.y;
        this.w = v.w; //!?
    }
    public Vector2f( float x, float y ) {
        this.x = x;
        this.y = y;
        this.w = 1.0f; //!?
    }
    public Vector2f( float x, float y, float w ) {
        this.x = x;
        this.y = y;
        this.w = w; //!?
    }
    public void translate( float tx, float ty ) {
        x += tx;
        y += ty;
    }
    public void scale( float sx, float sy ) {
        x *= sx;
        y *= sy;
    }
    public void rotate( float rad ) {
        float tmp = (float)( x * Math.cos( rad ) - y * Math.sin( rad ) );
        y = (float)( x * Math.sin( rad ) + y * Math.cos( rad ) );
        x = tmp;
    }
    public void shear( float sx, float sy ) {
        float tmp = x + sx * y;
        y = y + sy * x;
        x = tmp;
    }
}
```

VectorGraphicsExample 位于 javagames.transform 包中, 如图 3.2 所示, 它在屏幕上绘制一个多边形的飞船, 并且使用 Vector2f 类的方程式应用平移、旋转、缩放和切变。initialize()方法设置了多边形的飞船, 并且调用了 reset()。reset()方法负责将所有的变换值设置回其初始值。

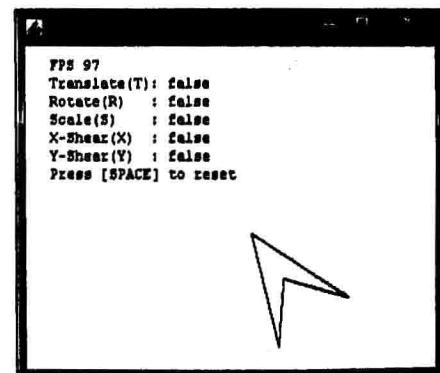


图 3.2 向量图形示例



有一些键对应到不同的变换操作。R 表示旋转，S 表示切变，T 表示平移，X 表示 x 切变，Y 表示 y 切变，空格表示将所有的值设置回其初始值。

这个多边形的对象在变换前复制到 world 列表中。通过这种方式，操作不是累积地进行，而是在每一帧都发生。这不仅使得最初的模型不变，还允许使用其多个副本。

每次变换操作，都是以相同的方式处理的。首先，检查映射到特定变换的 Boolean 值。如果它为真，那么会应用变换，检查边界，然后相应地调整步进值。在更新步进值之后，应用变换。注意，变换的顺序很重要。更改操作的顺序，将会改变屏幕上的结果。

在 render()方法内部，显示了所有变换标志的值。然后，使用图形对象的 drawLine() 方法绘制出变换后的多边形。终点和起点相连接，然后，每个点用于绘制从该点到前一个点的一条线段，最终得到一个闭合的多边形。

```
package javagames.transform;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javagames.util.*;
import javax.swing.*;

public class VectorGraphicsExample extends JFrame
    implements Runnable {
    private static final int SCREEN_W = 640;
    private static final int SCREEN_H = 480;
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private Vector2f[] polygon;
    private Vector2f[] world;
    private float tx, ty;
    private float vx, vy;
    private float rot, rotStep;
    private float scale, scaleStep;
    private float sx, sxStep;
    private float sy, syStep;
    private boolean doTranslate;
    private boolean doScale;
    private boolean doRotate;
    private boolean doXShear;
    private boolean doYShear;
```



```
public VectorGraphicsExample() {
}
protected void createAndShowGUI() {
    Canvas canvas = new Canvas();
    canvas.setSize( 640, 480 );
    canvas.setBackground( Color.BLACK );
    canvas.setIgnoreRepaint( true );
    getContentPane().add( canvas );
    setTitle( "Vector Graphics Example" );
    setIgnoreRepaint( true );
    pack();
    // Add key listeners
    keyboard = new KeyboardInput();
    canvas.addKeyListener( keyboard );
    // Add mouse listeners
    // For full screen : mouse = new RelativeMouseInput( this );
    mouse = new RelativeMouseInput( canvas );
    canvas.addMouseListener( mouse );
    canvas.addMouseMotionListener( mouse );
    canvas.addMouseWheelListener( mouse );
    setVisible( true );
    canvas.createBufferStrategy( 2 );
    bs = canvas.getBufferStrategy();
    canvas.requestFocus();
    gameThread = new Thread( this );
    gameThread.start();
}
public void run() {
    running = true;
    initialize();
    while( running ) {
        gameLoop();
    }
}
private void gameLoop() {
    processInput();
    processObjects();
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
```



```
try {
    g = bs.getDrawGraphics();
    g.clearRect( 0, 0, getWidth(), getHeight() );
    render( g );
} finally {
    if( g != null ) {
        g.dispose();
    }
}
} while( bs.contentsRestored() );
bs.show();
} while( bs.contentsLost() );
}

private void sleep( long sleep ) {
try {
    Thread.sleep( sleep );
} catch( InterruptedException ex ) { }
}

private void initialize() {
frameRate = new FrameRate();
frameRate.initialize();
polygon = new Vector2f[] {
    new Vector2f( 10, 0 ),
    new Vector2f(-10, 8 ),
    new Vector2f( 0, 0 ),
    new Vector2f( -10, -8 ),
};
world = new Vector2f[ polygon.length ];
reset();
}

private void reset() {
tx = SCREEN_W / 2;
ty = SCREEN_H / 2;
vx = vy = 2;
rot = 0.0f;
rotStep = (float)Math.toRadians( 1.0 );
scale = 1.0f;
scaleStep = 0.1f;
sx = sy = 0.0f;
sxStep = syStep = 0.01f;
doRotate = doScale = doTranslate = false;
doXShear = doYShear = false;
}

private void processInput() {
keyboard.poll();
```



```
mouse.poll();
if( keyboard.keyDownOnce( KeyEvent.VK_R ) ) {
    doRotate = !doRotate;
}
if( keyboard.keyDownOnce( KeyEvent.VK_S ) ) {
    doScale = !doScale;
}
if( keyboard.keyDownOnce( KeyEvent.VK_T ) ) {
    doTranslate = !doTranslate;
}
if( keyboard.keyDownOnce( KeyEvent.VK_X ) ) {
    doXShear = !doXShear;
}
if( keyboard.keyDownOnce( KeyEvent.VK_Y ) ) {
    doYShear = !doYShear;
}
if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
    reset();
}
}

private void processObjects() {
    // copy data...
    for( int i = 0; i < polygon.length; ++i ) {
        world[i] = new Vector2f( polygon[i] );
    }
    if( doScale ) {
        scale += scaleStep;
        if( scale < 1.0 || scale > 5.0 ) {
            scaleStep = -scaleStep;
        }
    }
    if( doRotate ) {
        rot += rotStep;
        if( rot < 0.0f || rot > 2*Math.PI ) {
            rotStep = -rotStep;
        }
    }
    if( doTranslate ) {
        tx += vx;
        if( tx < 0 || tx > SCREEN_W ) {
            vx = -vx;
        }
        ty += vy;
        if( ty < 0 || ty > SCREEN_H ) {
            vy = -vy;
        }
    }
}
```



```
        }
    }
    if( doXShear ) {
        sx += sxStep;
        if( Math.abs( sx ) > 2.0 ) {
            sxStep = -sxStep;
        }
    }
    if( doYShear ) {
        sy += syStep;
        if( Math.abs( sy ) > 2.0 ) {
            syStep = -syStep;
        }
    }
    for( int i = 0; i < world.length; ++i ) {
        world[i].shear( sx, sy );
        world[i].scale( scale, scale );
        world[i].rotate( rot );
        world[i].translate( tx, ty );
    }
}
private void render( Graphics g ) {
    g.setFont( new Font( "Courier New", Font.PLAIN, 12 ) );
    g.setColor( Color.GREEN );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );
    g.drawString( "Translate(T): " + doTranslate, 20, 35 );
    g.drawString( "Rotate(R) : " + doRotate, 20, 50 );
    g.drawString( "Scale(S) : " + doScale, 20, 65 );
    g.drawString( "X-Shear(X) : " + doXShear, 20, 80 );
    g.drawString( "Y-Shear(Y) : " + doYShear, 20, 95 );
    g.drawString( "Press [SPACE] to reset", 20, 110 );
    Vector2f S = world[ world.length - 1 ];
    Vector2f P = null;
    for( int i = 0; i < world.length; ++i ) {
        P = world[i];
        g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );
        S = P;
    }
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
```



```
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final VectorGraphicsExample app = new VectorGraphicsExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e.) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
```

## 3.2 使用极坐标

笛卡尔坐标（Cartesian coordinate）表示为一对 $(x,y)$ 值。极坐标（Polar coordinate）表示为一个角度值和一个距离值，如图 3.3 所示。

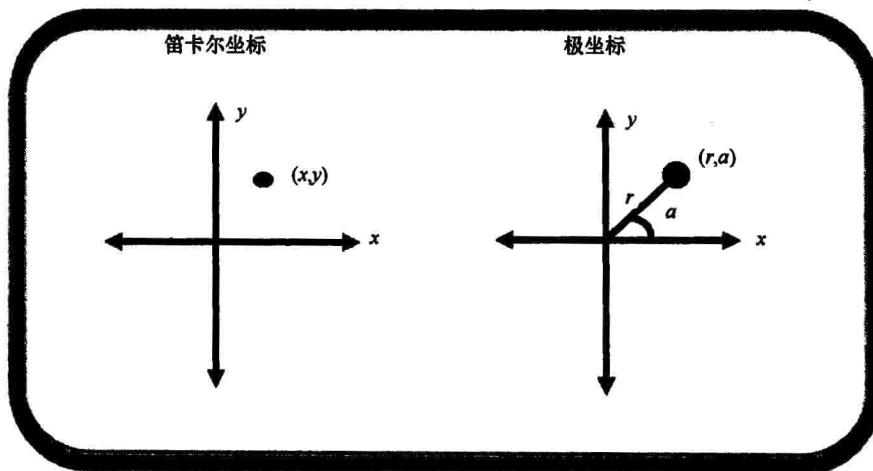


图 3.3 笛卡尔坐标和极坐标

极坐标的形式是 $(r,a)$ ，而不是 $(x,y)$ 对；其中  $r$  是距离（半径）， $a$  是角度。由于 Java 的三角函数使用弧度，因此所有的角度也都假设用弧度表示。在两种表示方式之间转换很容易：

```
Math.toRadians( double degrees )
Math.toDegrees( double radians )
```

要将笛卡尔坐标转换为极坐标，必须把点当做角来对待，使用三角数学来计算想要的值。如图 3.4 所示，距离可以看做是三角形的斜边，并使用毕达哥拉斯定律求得：

$$r^2 = a^2 + b^2$$
$$r = \sqrt{a^2 + b^2}$$

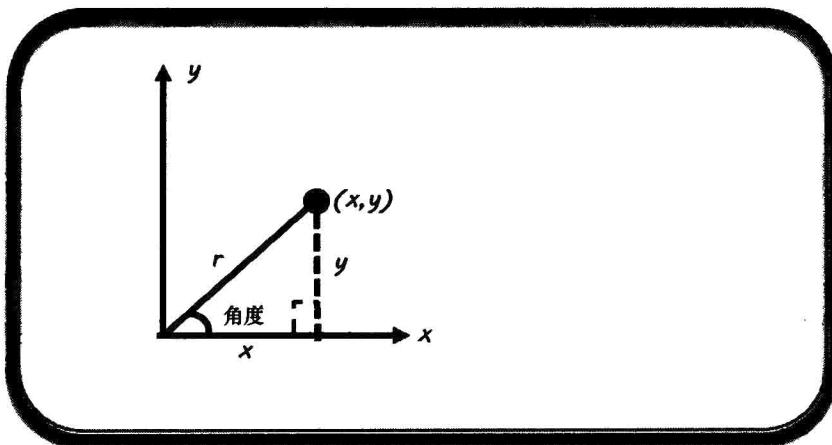


图 3.4 笛卡尔坐标系转换为极坐标系

可以使用正切函数求得角度：

$$\theta = \tan \frac{y}{x}$$

将极坐标转换为 $(x,y)$ 坐标值对也很容易：

$$x = r \cos \theta$$

$$y = r \sin \theta$$

可以根据前面的定义，得到前面所给出的旋转公式。从开始角度旋转一定的弧度数，等同于给初始角度加上想要的额外旋转角度。换句话说，要将一个点旋转一定的弧度数，可以用如下公式表示：

$$x' = r \cos(\theta + \varphi)$$
$$y' = r \sin(\theta + \varphi)$$



所采用的三角函数：

$$\begin{aligned}\sin(\theta + \varphi) &= \sin \theta \cos \varphi + \cos \theta \sin \varphi \\ \cos(\theta + \varphi) &= \cos \theta \cos \varphi - \sin \theta \sin \varphi\end{aligned}$$

将前面的公式，替换为如下的新的表示方式：

$$\begin{aligned}x' &= r(\cos \theta \cos \varphi - \sin \theta \sin \varphi) \\ y' &= r(\sin \theta \cos \varphi + \cos \theta \sin \varphi)\end{aligned}$$

将  $r$  值代入以得到：

$$\begin{aligned}x' &= r \cos \theta \cos \varphi - r \sin \theta \sin \varphi \\ y' &= r \sin \theta \cos \varphi + r \cos \theta \sin \varphi\end{aligned}$$

记住如下的公式：

$$\begin{aligned}x &= r \cos \theta \\ y &= r \sin \theta\end{aligned}$$

代入最初的  $x$  和  $y$  值，得到如下形式：

$$\begin{aligned}x' &= x \cos \varphi - y \sin \varphi \\ y' &= y \cos \varphi + x \sin \varphi\end{aligned}$$

前面这个公式表示 Vector2f 类中的一个坐标的旋转。PolarCoordinateExample 位于

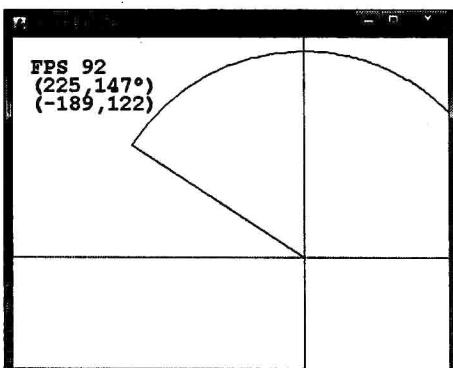


图 3.5 极坐标示例

javagames.transform 包下，如图 3.5 所示，它将一个笛卡尔坐标转换为一个极坐标，并再次转换回来。coord 变量保存了鼠标所在的屏幕位置，距屏幕中心有一定的偏移量。

转换在 render()方法中进行。 $(cx, cy)$  坐标表示屏幕的中心，而  $(px, py)$  坐标表示当前鼠标位置距离屏幕中心的偏移量。由于  $y$  值在屏幕坐标向下方向是正值，因此  $py$  变量的计算方法有所不同。

```
int px = coord.x - cx;  
int py = cy - coord.y;
```

从  $(px, py)$  值到极坐标值的转换中，用  $r$  变量来存储距离，用  $rad$  变量来存储角度的弧度数。Math.atan2()方法返回的角度在  $(-180, 180)$  之间，而角度数需要调整为  $(0, 360)$  之间。

```
if( degrees < 0 ) {  
    degrees = 360 - degrees;  
}
```



(sx,sy)变量保存了从极坐标向笛卡尔坐标的转换。

```
double sx = r * Math.cos(rad);
double sy = r * Math.sin(rad);
```

不同的值显示在屏幕上，以便可以检查转换的值。起始值和转换后的值应该是相同的。最后，用图形对象的 drawArc()方法来显示曲线。该方法接受一个左上角的点、弧形的边界矩形的宽度和高度，以及用角度数（而不是弧度）表示的开始角度和结束角度。

在使用 drawArc()方法之前，确保将弧度转换为角度。



```
package javagames.transform;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javagames.util.*;
import javax.swing.*;

public class PolarCoordinateExample extends JFrame
    implements Runnable {
    private static final int SCREEN_W = 640;
    private static final int SCREEN_H = 480;
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private Point coord;
    public PolarCoordinateExample() {
    }
    protected void createAndShowGUI() {
        Canvas canvas = new Canvas();
        canvas.setSize( SCREEN_W, SCREEN_H );
        canvas.setBackground( Color.BLACK );
        canvas.setIgnoreRepaint( true );
        getContentPane().add( canvas );
        setTitle( "Polar Coordinate Example" );
        setIgnoreRepaint( true );
        pack();
        // Add key listeners
    }
}
```



```
keyboard = new KeyboardInput();
canvas.addKeyListener( keyboard );
// Add mouse listeners
// For full screen : mouse = new RelativeMouseInput( this );
mouse = new RelativeMouseInput( canvas );
canvas.addMouseListener( mouse );
canvas.addMouseMotionListener( mouse );
canvas.addMouseWheelListener( mouse );
setVisible( true );
canvas.createBufferStrategy( 2 );
bs = canvas.getBufferStrategy();
canvas.requestFocus();
gameThread = new Thread( this );
gameThread.start();
}
public void run() {
    running = true;
    initialize();
    while( running ) {
        gameLoop();
    }
}
private void gameLoop() {
    processInput();
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
            try {
                g = bs.getDrawGraphics();
                g.clearRect( 0, 0, getWidth(), getHeight() );
                render( g );
            } finally {
                if( g != null ) {
                    g.dispose();
                }
            }
        }
    } while( bs.contentsRestored() );
    bs.show();
} while( bs.contentsLost() );
```



```
}

private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}

private void initialize() {
    frameRate = new FrameRate();
    frameRate.initialize();
    coord = new Point();
}

private void processInput() {
    keyboard.poll();
    mouse.poll();
    coord = mouse.getPosition();
}

private void render( Graphics g ) {
    g.setFont( new Font( "Courier New", Font.BOLD, 24 ) );
    g.setColor( Color.GREEN );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 40 );
    int cx = SCREEN_W / 2;
    int cy = SCREEN_H / 2;
    g.setColor( Color.GRAY );
    g.drawLine( 0, cy, SCREEN_W, cy );
    g.drawLine( cx, 0, cx, SCREEN_H );
    g.setColor( Color.GREEN );
    g.drawLine( cx, cy, coord.x, coord.y );
    int px = coord.x - cx;
    int py = cy - coord.y;
    double r = Math.sqrt( px*px + py*py );
    double rad = Math.atan2( py, px );
    double degrees = Math.toDegrees( rad );
    if( degrees < 0 ) {
        degrees = 360 + degrees;
    }
    double sx = r * Math.cos( rad );
    double sy = r * Math.sin( rad );
    String polar = String.format( "(%.0f,%.0f\u00b0)", r, degrees );
    g.drawString( polar, 20, 60 );
    String cart = String.format( "(%.0f,%.0f)", sx, sy );
    g.drawString( cart, 20, 80 );
    g.setColor( Color.WHITE );
```



```
g.drawString( String.format( "(%s,%s)", px, py ), coord.x, coord.y );
g.setColor( Color.BLUE );
g.drawArc(
    (int)(cx - r), (int)(cy - r),
    (int)(2*r), (int)(2*r), 0, (int)degrees
);
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final PolarCoordinateExample app = new PolarCoordinateExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
}
```

### 3.3 理解点和向量

如图 3.6 所示，点和向量之间的区别容易令人混淆。2D 空间中的一个点表示为一个  $(x,y)$  对。一个向量可以表示为两个坐标对：一个  $(x,y)$  坐标表示向量的起点，另一个坐标表示终点。然而，第一个坐标往往隐式地表示为  $(0,0)$ ，并且在代码表示中省略掉。

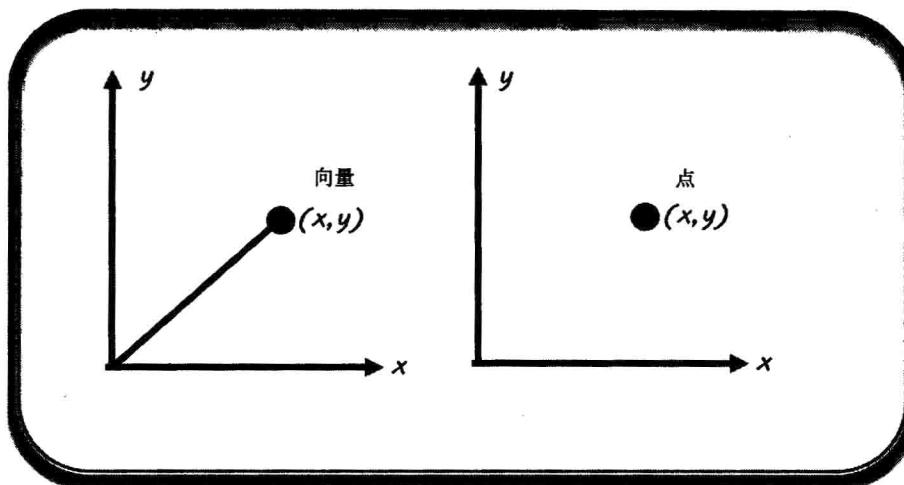


图 3.6 点和向量的相似性



由于向量和点可以以相同的方式来表示，人们可能很难理解其区别。加之  
一些懒惰的程序员（像我这样的），可能对于向量和点都使用一个向量类，  
这让情况变得更加容易令人混淆。

把点想象为一座城市，而把向量想象成开车的方向，这可能会有所帮助。

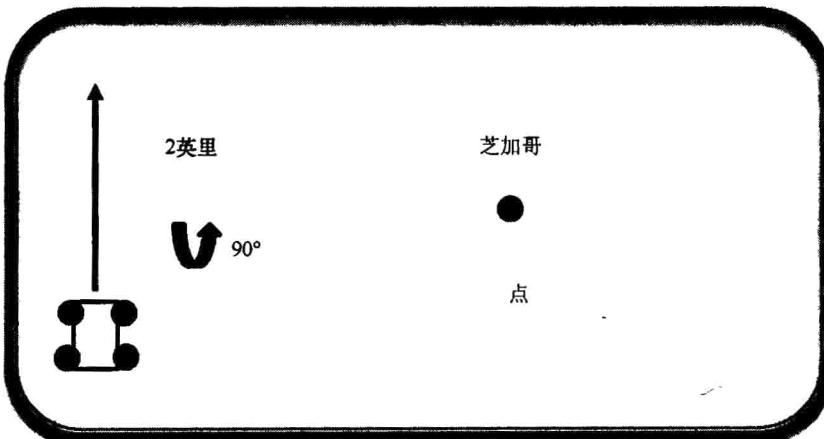


图 3.7 开车方向、位置、向量和点

如图 3.7 所示，芝加哥市是一个点。把车掉转 90 度向左，并且直线开出 2 英里，这是  
一个向量。从地球上的任何位置开始，向量的开车方向将会把汽车放置到另外的某个地方。

由于汽车可以从任何位置开始，因此只有一个向量还没有足够的信息来确定一个位置。

可以对向量和点执行各种数学操作。例如，把这些开车方向和一个起始点（例如，芝加哥）相加，将会得到一个新的点，或者一个新的位置。可以将向量和另一个向量相加或者相减，但是，由于它们可以从任何地方开始，因此结果也是可以从任何地方开始的另一个向量，如图 3.8 所示。

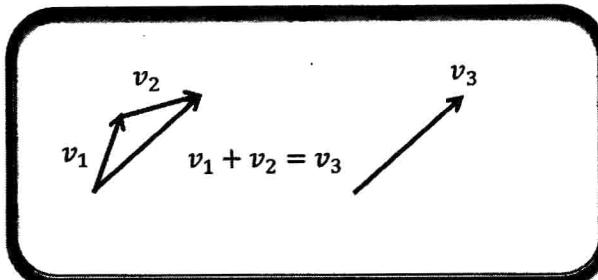


图 3.8 向量相加

将两个点相加是没有意义的：

Chicago + New York = ???

当初次学习向量和点的时候，会导致很多的混淆。有些事情在数学上是正确的，并且随后有些事情你需要在代码中使用向量和点去完成，以使游戏能够工作。正如前面所讨论的，要使用一个矩阵来操作向量，需要加一个额外的 w 值。这就是可以通过 w 值来区分一个向量和一个点的原因。如果 w 等于 1，平移可以进行。当 w 值等于 0 的时候，平移无效。点和向量都可以使用相同的数据结构来表示，只要将 w 值从 1 改为 0 就可以做到。

## 3.4 使用矩阵变换

前面所有的变换都可以使用一个矩阵来应用。有表示平移、旋转、缩放和切变的矩阵。这些不同的矩阵对象可以相乘，将所有不同的操作组合到一个单个的矩阵中，然后可以使它一次性变换所有的点。矩阵相乘和将两个数相乘不同，将一个向量和 3 个元素相乘可以得到拥有  $3 \times 3$  个元素的一个矩阵，理解这一点是很重要的。

将两个不同大小的矩阵相乘，其内部的大小必须匹配。所得到的矩阵将拥有外围矩阵的大小。换句话说， $A \times B * B \times C = A \times C$ 。内部的 B 的数值必须一致。如下是有效的矩阵大小：



- $4 \times 2 * 2 \times 3 = 4 \times 3$
- $3 \times 3 * 3 \times 5 = 3 \times 5$
- $1 \times 2 * 2 \times 7 = 1 \times 7$

这也是需要额外的 w 值的原因之一。 $1 \times 2 * 3 \times 3$  的矩阵操作是无效的。

此外，向量和矩阵的顺序必须是正确的，否则的话，矩阵操作将会是无效的。

- $3 \times 3 * 1 \times 3 = \text{Error, vector can't be a } 1 \times 3$
- $3 \times 3 * 3 \times 1 = \text{Good}$

另一个重要之处在于，矩阵相乘是不可以互换的。换句话说：

$M1 * M2 != M2 * M1$

调换相乘的顺序将会改变结果，这一点和其他的乘法是不同的。平移然后旋转，所得到的结果和旋转然后平移是不同的。记住这一点很重要。将一个矩阵和另一个矩阵相乘的时候，如果互换两个矩阵，不会得到相同的结果，尽管可以通过选择值的方式迫使结果相同。因此，互换向量和矩阵以使得乘法有效，这不是一个好主意：

`1x3 vector * 3x3 matrix != 3x3 matrix * 3x1 vector`

当操作矩阵的时候，这将会引发下一个重要的思考，即如何正确使用行主序格式和列主序格式。

## 3.5 行主序矩阵和列主序矩阵

如下的变换矩阵是列主序矩阵：

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

向量(tx, ty, 1)是平移矩阵，并且从上到下出现，占据了矩阵的一列。

如下的矩阵变换是行主序矩阵：

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

注意，变换向量现在跑到了底部，在一行上。为什么这很重要？第一个原因是，这个



平移矩阵的值处在不同的位置，因此，值匹配不对将会导致错误。第二个原因是，正如前面所述， $M1 * M2 \neq M2 * M1$ 。这意味着，顺序很重要。这个问题涉及操作的顺序。

让  $C =$  矩阵的一个连接。如果你缩放 (S)、旋转 (R) 和平移 (T)，最终有两种不同的方法来编写代码：

```
C = S * R * T  
C = T * R * S
```

你会像下面这么做吗？

```
C = T.mul(R).mul(S);
```

或者是用这种方法？

```
C = S.mul(R).mul(T);
```

答案是，任何一种方法都是正确的，这取决于你使用哪种矩阵顺序。如果矩阵以行主序的形式存储，那么，矩阵连接从左向右进行。然而，如果使用列主序的形式，矩阵连接从右向左执行。记住， $M * N \neq N * M$ ，你可以看到，很容易将矩阵放错位置，或者以错误的顺序来做事情。



在编写本章的第一个版本的时候，我没有太多地介绍行主序和列主序。  
在第一次审阅之后，我意识到已经交换了一些代码。我在用列主序来对行主序的矩阵做事情。在修正了这一错误并且重新测试了其他的内容之后，我添加了这一小节。

甚至更为复杂的是，很容易写出如下的代码：

```
Matrix3x3f mat = Matrix3x3f.translate( tx, ty );  
mat = mat.mul( Matrix3x3f.rotate( rad ) );
```

这会执行  $C = T * R$ 。这很容易写成：

```
Matrix3x3f mat = Matrix3x3f.translate( tx, ty );  
mat = Matrix3x3f.rotate( rad ).mul( mat );
```

这会执行  $C = R * T$ 。但是，除非指定了矩阵的行/列顺序，否则没有办法理解旋转是发生在平移之前还是平移之后。这可能很容易令人混淆，因此，确保你理解会发生什么事情。

从行主序改为列主序，并不只是涉及使用一个不同的旋转矩阵。向量\*矩阵的顺序互换了。如果使用行主序，那么，向量是行，这就是一个  $1 \times 3$  的矩阵。如果使用列主序，那么，向量就是列，这是一个  $3 \times 1$  的矩阵。行主序是  $1 \times 3 * 3 \times 3$ ，但列主序是  $3 \times 3 * 3 \times 1$ 。我必须更新代码，以便把列主序切换为行主序向量相乘，并且切换相乘的顺序。



如果在所有可用的教程中搜索旋转矩阵，那么会发现有些不同。一些库、教程和文档介绍行主序的矩阵变换，而另一些则介绍列主序。有一种非常流行的 3D 图形库，OpenGL，它把矩阵在内存中存储为数据的一个单个数组，但是，一些文档则是以列主序的格式来展示值。我也看到过很多 DirectX 文档使用行主序格式。



我记得初次学习矩阵旋转的时候，我阅读了 Web 站点上的教程，并且遇到了这样一个论坛，两拨人在那里就旋转矩阵展开争论。每一方都宣称别人的负号放错了地方。

```
[cos(a), -sin(a), 0] [cos(a), sin(a), 0]  
[sin(a), cos(a), 0] [-sin(a), cos(a), 0]  
[0, 0, 1] [0, 0, 1]
```

当然，两方都是正确的，只不过一方使用行主序的表示法，而另一方使用列主序的表示法。已经警告过你了。

讨论完你第一次学习所有这些内容时将会面临的所有通用性问题之后，我必须介绍一下最复杂的部分：将两个矩阵相乘。下一节将介绍所有有趣的数学内容所使用的矩阵类。

## 3.6 理解 Matrix3x3f 类

将矩阵相加和相减的代码很简单，即两个矩阵的每个索引的值彼此相加或相减，然后放入到同样的索引位置。只有具有相同宽度和高度的矩阵才能够相加减，并且你可能发现实际上不会需要将一个矩阵和另一个矩阵相加或相减。

乘法的情况变得复杂起来。第一个矩阵的第一行和第二个矩阵的所有 3 列相乘。然后，第一个矩阵的中间一行和第二矩阵的所有 3 列相乘。最后，第一个矩阵的最后一行，和第二矩阵的所有 3 列相乘。这种模式可以展示为如下所示的两种不同方式：

```
[a, b, c] [1, 2, 3] .  
A [d, e, f] B [4, 5, 6] = C  
[g, h, i] [7, 8, 9]  
  
[a*1 + b*4 + c*7, a*2 + b*5 + c*8, a*3 + b*6 + c*9]  
C [d*1 + e*4 + f*7, d*2 + e*5 + f*8, d*3 + e*6 + f*9]  
[g*1 + h*4 + i*7, g*2 + h*5 + i*8, g*3 + h*6 + i*9]
```



```
C[0,0] = a*1 + b*4 + c*7  
C[0,1] = a*2 + b*5 + c*8  
C[0,2] = a*3 + b*6 + c*9  
C[1,0] = d*1 + e*4 + f*7  
C[1,1] = d*2 + e*5 + f*8  
C[1,2] = d*3 + e*6 + f*9  
C[2,0] = g*1 + h*4 + i*7  
C[2,1] = g*2 + h*5 + i*8  
C[2,2] = g*3 + h*6 + i*9
```

将一个向量和 3 个点 [x, y, w] 相乘，这基本上和矩阵相乘相同，只不过这里只有一行，而不是三行。一个行主序的、 $1 \times 3 \times 3 \times 3$  的向量和矩阵相乘，如下所示：

```
[m00, m01, m02]  
V[x, y, w] * [m10, m11, m12]  
[m20, m21, m22]
```

```
Vx = m00 * x + m10 * y + m20 * w  
Vy = m01 * x + m11 * y + m21 * w  
Vw = m02 * x + m12 * y + m22 * w
```

如果使用列主序的矩阵，那么，向量是一列，而不是一行。一个  $3 \times 3 * 3 \times 1$  的列主序的矩阵和向量相乘，如下所示：

```
[m00, m01, m02]      [x]  
[m10, m11, m12] * V[y]  
[m20, m21, m22]      [w]  
  
Vx = m00 * x + m01 * y + m02 * w  
Vy = m10 * x + m11 * y + m12 * w  
Vw = m20 * x + m21 * y + m22 * w
```



如果你选择将行主序矩阵切换为列主序矩阵的话，代码的所有位置都不再相同了，要确保理解这一点。这很容易会漏掉某些内容（相信我）。

如下的变换把乘法展开来，以展示发生了什么。对于 0 矩阵，我省略了乘法，因为用什么乘以 0 都没关系，结果总是 0。

然而，恒等矩阵是一种特殊的矩阵，它不会改变任何矩阵的值。它等同于矩阵和 1 相乘的结果。

```
[1, 0, 0]      [x]  
Identity [0, 1, 0] * V [y]  
[0, 0, 1]      [w]
```



$$\begin{aligned}Vx &= 1*x + 0*y + 0*w = x \\Vy &= 0*x + 1*y + 0*w = y \\Vw &= 0*x + 0*y + 1*w = w\end{aligned}$$

缩放矩阵就像预期的那样工作：

$$\begin{aligned}&\begin{bmatrix} sx, 0, 0 \end{bmatrix} \quad [x] \\Scale \begin{bmatrix} 0, sy, 0 \end{bmatrix} * V[y] \\&\begin{bmatrix} 0, 0, 1 \end{bmatrix} \quad [w]\end{aligned}\begin{aligned}Vx &= sx*x + 0*y + 0*w = sx*x \\Vy &= 0*x + sy*y + 0*w = sy*y \\Vw &= 0*x + 0*y + 1*w = w\end{aligned}$$

只有在  $w=1$  的时候，平移矩阵才有效。如果  $w=0$ ，那么平移操作将不会应用：

$$\begin{aligned}&\begin{bmatrix} 1, 0, tx \end{bmatrix} \quad [x] \\Translate \begin{bmatrix} 0, 1, ty \end{bmatrix} * V[y] \\&\begin{bmatrix} 0, 0, 1 \end{bmatrix} \quad [w]\end{aligned}\begin{aligned}Vx &= 1*x + 0*y + tx*w = x + tx*w \\Vy &= 0*x + 1*y + ty*w = y + ty*w \\Vw &= 0*x + 0*y + 1*w = w\end{aligned}$$

正如前面所介绍的，添加了  $w$  值使变换有效。如果没有  $w$  值，那么，这个变换是不可能的。此外，注意， $w=0$  会导致变换值为 0，这应该是变换一个向量的正确的值。

切变矩阵如下所示：

$$\begin{aligned}&\begin{bmatrix} 1, sx, 0 \end{bmatrix} \quad [x] \\Shear \begin{bmatrix} sy, 1, 0 \end{bmatrix} * V[y] \\&\begin{bmatrix} 0, 0, 1 \end{bmatrix} \quad [w]\end{aligned}\begin{aligned}Vx &= 1*x + sx*y + 0*w = x + sx*y \\Vy &= sy*x + 1*y + 0*w = y + sy*x \\Vw &= 0*x + 0*y + 1*w = w\end{aligned}$$

最后，旋转矩阵如下所示：

$$\begin{aligned}&\begin{bmatrix} \cos(a), -\sin(a), 0 \end{bmatrix} \quad [x] \\Rotation \begin{bmatrix} \sin(a), \cos(a), 0 \end{bmatrix} * V[y] \\&\begin{bmatrix} 0, 0, 1 \end{bmatrix} \quad [w]\end{aligned}\begin{aligned}Vx &= x*\cos(a) + -y*\sin(a) + 0*w = x*\cos(a) - y*\sin(a) \\Vy &= x*\sin(a) + y*\cos(a) + 0*w = x*\sin(a) + y*\cos(a) \\Vw &= 0*x + 0*y + 1*w = w\end{aligned}$$

这也和前面给出的 `vector` 类中的旋转代码一致。



```
package javagames.util;

public class Matrix3x3f {
    private float[][] m = new float[ 3 ][ 3 ];
    public Matrix3x3f() {
    }
    public Matrix3x3f( float[][] m ) {
        setMatrix( m );
    }
    public Matrix3x3f add( Matrix3x3f m1 ) {
        return new Matrix3x3f( new float[][] {
            { this.m[ 0 ][ 0 ] + m1.m[ 0 ][ 0 ],
                this.m[ 0 ][ 1 ] + m1.m[ 0 ][ 1 ],
                this.m[ 0 ][ 2 ] + m1.m[ 0 ][ 2 ] },
            { this.m[ 1 ][ 0 ] + m1.m[ 1 ][ 0 ],
                this.m[ 1 ][ 1 ] + m1.m[ 1 ][ 1 ],
                this.m[ 1 ][ 2 ] + m1.m[ 1 ][ 2 ] },
            { this.m[ 2 ][ 0 ] + m1.m[ 2 ][ 0 ],
                this.m[ 2 ][ 1 ] + m1.m[ 2 ][ 1 ],
                this.m[ 2 ][ 2 ] + m1.m[ 2 ][ 2 ] } } );
    }

    public Matrix3x3f sub( Matrix3x3f m1 ) {
        return new Matrix3x3f( new float[][] {
            { this.m[ 0 ][ 0 ] - m1.m[ 0 ][ 0 ],
                this.m[ 0 ][ 1 ] - m1.m[ 0 ][ 1 ],
                this.m[ 0 ][ 2 ] - m1.m[ 0 ][ 2 ] },
            { this.m[ 1 ][ 0 ] - m1.m[ 1 ][ 0 ],
                this.m[ 1 ][ 1 ] - m1.m[ 1 ][ 1 ],
                this.m[ 1 ][ 2 ] - m1.m[ 1 ][ 2 ] },
            { this.m[ 2 ][ 0 ] - m1.m[ 2 ][ 0 ],
                this.m[ 2 ][ 1 ] - m1.m[ 2 ][ 1 ],
                this.m[ 2 ][ 2 ] - m1.m[ 2 ][ 2 ] } } );
    }

    public Matrix3x3f mul( Matrix3x3f m1 ) {
        return new Matrix3x3f( new float[][] {
            { this.m[ 0 ][ 0 ] * m1.m[ 0 ][ 0 ] // *****
+ this.m[ 0 ][ 1 ] * m1.m[ 1 ][ 0 ] // M[0,0]
+ this.m[ 0 ][ 2 ] * m1.m[ 2 ][ 0 ], // *****
                this.m[ 0 ][ 0 ] * m1.m[ 0 ][ 1 ] // *****
+ this.m[ 0 ][ 1 ] * m1.m[ 1 ][ 1 ] // M[0,1]
+ this.m[ 0 ][ 2 ] * m1.m[ 2 ][ 1 ], // *****
                this.m[ 0 ][ 0 ] * m1.m[ 0 ][ 2 ] // *****
```



### 3.6 理解 Matrix



```
+ this.m[ 0 ][ 1 ] * m1.m[ 1 ][ 2 ] // M[0,2]
+ this.m[ 0 ][ 2 ] * m1.m[ 2 ][ 2 ] },// *****
{ this.m[ 1 ][ 0 ] * m1.m[ 0 ][ 0 ] // *****
+ this.m[ 1 ][ 1 ] * m1.m[ 1 ][ 0 ] // M[1,0]
+ this.m[ 1 ][ 2 ] * m1.m[ 2 ][ 0 ], // *****
    this.m[ 1 ][ 0 ] * m1.m[ 0 ][ 1 ] // *****
+ this.m[ 1 ][ 1 ] * m1.m[ 1 ][ 1 ] // M[1,1]
+ this.m[ 1 ][ 2 ] * m1.m[ 2 ][ 1 ], // *****
    this.m[ 1 ][ 0 ] * m1.m[ 0 ][ 2 ] // *****
+ this.m[ 1 ][ 1 ] * m1.m[ 1 ][ 2 ] // M[1,2]
+ this.m[ 1 ][ 2 ] * m1.m[ 2 ][ 2 ],// *****
{ this.m[ 2 ][ 0 ] * m1.m[ 0 ][ 0 ] // *****
+ this.m[ 2 ][ 1 ] * m1.m[ 1 ][ 0 ] // M[2,0]
+ this.m[ 2 ][ 2 ] * m1.m[ 2 ][ 0 ], // *****
    this.m[ 2 ][ 0 ] * m1.m[ 0 ][ 1 ] // *****
+ this.m[ 2 ][ 1 ] * m1.m[ 1 ][ 1 ] // M[2,1]
+ this.m[ 2 ][ 2 ] * m1.m[ 2 ][ 1 ], // *****
    this.m[ 2 ][ 0 ] * m1.m[ 0 ][ 2 ] // *****
+ this.m[ 2 ][ 1 ] * m1.m[ 1 ][ 2 ] // M[2,2]
+ this.m[ 2 ][ 2 ] * m1.m[ 2 ][ 2 ] } // *****
});
}

public void setMatrix( float[][] m ) {
    this.m = m;
}
public static Matrix3x3f zero() {
    return new Matrix3x3f( new float[][] {
        { 0.0f, 0.0f, 0.0f },
        { 0.0f, 0.0f, 0.0f },
        { 0.0f, 0.0f, 0.0f }
    });
}
public static Matrix3x3f identity() {
    return new Matrix3x3f( new float[][] {
        { 1.0f, 0.0f, 0.0f },
        { 0.0f, 1.0f, 0.0f },
        { 0.0f, 0.0f, 1.0f }
    });
}
public static Matrix3x3f translate( Vector2f v ) {
    return translate( v.x, v.y );
}
```



```
public static Matrix3x3f translate( float x, float y ) {
    return new Matrix3x3f( new float[][] {
        { 1.0f, 0.0f, 0.0f },
        { 0.0f, 1.0f, 0.0f },
        { x, y, 1.0f }
    });
}
public static Matrix3x3f scale( Vector2f v ) {
    return scale( v.x, v.y );
}
public static Matrix3x3f scale( float x, float y ) {
    return new Matrix3x3f( new float[][] {
        { x, 0.0f, 0.0f },
        { 0.0f, y, 0.0f },
        { 0.0f, 0.0f, 1.0f }
    });
}
public static Matrix3x3f shear( Vector2f v ) {
    return shear( v.x, v.y );
}

public static Matrix3x3f shear( float x, float y ) {
    return new Matrix3x3f( new float[][] {
        { 1.0f, y, 0.0f },
        { x, 1.0f, 0.0f },
        { 0.0f, 0.0f, 1.0f }
    });
}
public static Matrix3x3f rotate( float rad ) {
    return new Matrix3x3f( new float[][] {
        { (float) Math.cos(rad), (float) Math.sin( rad ), 0.0f },
        { (float)-Math.sin(rad), (float) Math.cos( rad ), 0.0f },
        { 0.0f, 0.0f, 1.0f }
    });
}
public Vector2f mul( Vector2f vec ) {
    return new Vector2f(
        vec.x * this.m[ 0 ][ 0 ] // 
        + vec.y * this.m[ 1 ][ 0 ] // v.x
        + vec.w * this.m[ 2 ][ 0 ], //
        vec.x * this.m[ 0 ][ 1 ] // 
        + vec.y * this.m[ 1 ][ 1 ] // v.y
        + vec.w * this.m[ 2 ][ 1 ], //
    );
}
```



```
    vec.x * this.m[ 0 ][ 2 ] //  
+ vec.y * this.m[ 1 ][ 2 ] // v.w  
+ vec.w * this.m[ 2 ][ 2 ] //  
);  
}  
@Override  
public String toString() {  
    StringBuilder buf = new StringBuilder();  
    for( int i = 0; i < 3; ++i ) {  
        buf.append( "[" );  
        buf.append( m[i][0] );  
        buf.append( ",\t" );  
        buf.append( m[i][1] );  
        buf.append( ",\t" );  
        buf.append( m[i][2] );  
        buf.append( ")\n" );  
    }  
    return buf.toString();  
}  
}
```

MatrixMultiplicationExample 示例位于 javagames.transform 包中，如图 3.9 所示，它展示了使用矩阵连接在对象上的变换操作。矩阵之所以成为一个如此强大的工具，是因为诸如变换和旋转这样的操作可以存储到单个的矩阵中，然后一次性地应用于对象。尽管这对于单个对象来说并非是必须的，但如果需要变换成百上千个多边形的话，所节省的时间相当可观。



图 3.9 矩阵相乘示例



为了有趣，这个示例还创建了一些随机的星形，可以使用空格键来打开或关闭它。

以恒等矩阵为起点，然后与一个平移矩阵相乘，将其移动到屏幕的中央。然后，Sun 向量和矩阵相乘，将 Sun 的中心放置在屏幕的中央。尽管这只是一个单个的点，但一个完整的多边形模型也可以通过矩阵相乘来变换其全部的坐标。

将地球轨道的路径绘制成一个白色的圆，然后，绘制地球，它绕着太阳旋转。注意，这使用了相同的矩阵。将地球的旋转和平移连接起来，只需要增加一次旋转和平移，就可以让地球围绕太阳旋转起来。这个示例和前一个示例的不同之处很重要。参见图 3.10 和图 3.11 来了解先旋转和先平移之间的区别。

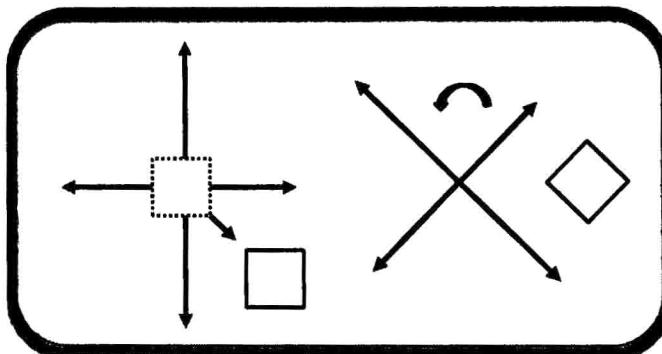


图 3.10 先平移再旋转

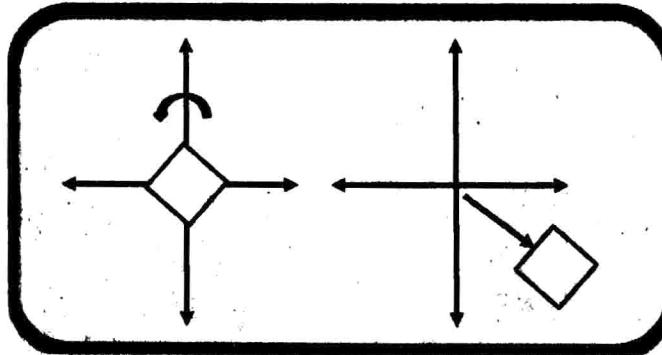


图 3.11 先旋转再平移

还是使用相同的矩阵，另一个圆表示月球围绕地球旋转的轨道。它们也都围绕太阳旋转。使用矩阵相乘，这些都可以做到而且很容易做到。请确保研究代码，以了解最终的意义。



```
package javagames.transform;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.util.Random;
import javagames.util.*;
import javax.swing.*;

public class MatrixMultiplyExample extends JFrame implements Runnable {
    private static final int SCREEN_W = 640;
    private static final int SCREEN_H = 480;
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private float earthRot, earthDelta;
    private float moonRot, moonDelta;
    private boolean showStars;
    private int[] stars;
    private Random rand = new Random();
    public MatrixMultiplyExample() {
    }
    protected void createAndShowGUI() {
        Canvas canvas = new Canvas();
        canvas.setSize( SCREEN_W, SCREEN_H );
        canvas.setBackground( Color.BLACK );
        canvas.setIgnoreRepaint( true );
        getContentPane().add( canvas );
        setTitle( "Matrix Multiply Example" );
        setIgnoreRepaint( true );
        pack();
        // Add key listeners
        keyboard = new KeyboardInput();
        canvas.addKeyListener( keyboard );
        // Add mouse listeners
        // For full screen : mouse = new RelativeMouseInput( this );
        mouse = new RelativeMouseInput( canvas );
        canvas.addMouseListener( mouse );
        canvas.addMouseMotionListener( mouse );
        canvas.addMouseWheelListener( mouse );
        setVisible( true );
        canvas.createBufferStrategy( 2 );
    }
}
```



```
bs = canvas.getBufferStrategy();
canvas.requestFocus();
gameThread = new Thread( this );
gameThread.start();
}
public void run() {
    running = true;
    initialize();
    while( running ) {
        gameLoop();
    }
}
private void gameLoop() {
    processInput();
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
            try {
                g = bs.getDrawGraphics();
                g.clearRect( 0, 0, getWidth(), getHeight() );
                render( g );
            } finally {
                if( g != null ) {
                    g.dispose();
                }
            }
        }
    } while( bs.contentsRestored() );
    bs.show();
} while( bs.contentsLost() );
}
private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}
private void initialize() {
    frameRate = new FrameRate();
    frameRate.initialize();
    earthDelta = (float)Math.toRadians( 0.5 );
```



```
moonDelta = (float)Math.toRadians( 2.5 );
showStars = true;
stars = new int[ 1000 ];
for( int i = 0; i < stars.length - 1; i += 2 ) {
    stars[i] = rand.nextInt( SCREEN_W );
    stars[i+1] = rand.nextInt( SCREEN_H );
}
}
private void processInput() {
    keyboard.poll();
    mouse.poll();
    if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
        showStars = !showStars;
    }
}
private void render( Graphics g ) {
    g.setColor( Color.GREEN );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );
    g.drawString( "Press [SPACE] to toggle stars", 20, 35 );
    if( showStars ) {
        g.setColor( Color.WHITE );
        for( int i = 0; i < stars.length - 1; i += 2 ) {
            g.fillRect( stars[i], stars[i+1], 1, 1 );
        }
    }
    // draw the sun...
    Matrix3x3f sunMat = Matrix3x3f.identity();
    sunMat = sunMat.mul(
        Matrix3x3f.translate( SCREEN_W / 2, SCREEN_H / 2 )
    );
    Vector2f sun = sunMat.mul( new Vector2f() );
    g.setColor( Color.YELLOW );
    g.fillOval( (int)sun.x - 50, (int)sun.y - 50, 100, 100 );
    // draw Earth's Orbit
    g.setColor( Color.WHITE );
    g.drawOval( (int)sun.x - SCREEN_W / 4, (int)sun.y - SCREEN_W / 4,
               SCREEN_W / 2, SCREEN_W / 2 );
    // draw the Earth
    Matrix3x3f earthMat = Matrix3x3f.translate( SCREEN_W / 4, 0 );
    earthMat = earthMat.mul( Matrix3x3f.rotate( earthRot ) );
    earthMat = earthMat.mul( sunMat );
    earthRot += earthDelta;
```



```
Vector2f earth = earthMat.mul( new Vector2f() );
g.setColor( Color.BLUE );
g.fillOval( (int)earth.x - 10, (int)earth.y - 10, 20, 20 );
// draw the Moon
Matrix3x3f moonMat = Matrix3x3f.translate( 30, 0 );
moonMat = moonMat.mul( Matrix3x3f.rotate( moonRot ) );
moonMat = moonMat.mul( earthMat );
moonRot += moonDelta;
Vector2f moon = moonMat.mul( new Vector2f() );
g.setColor( Color.LIGHT_GRAY );
g.fillOval( (int)moon.x - 5, (int)moon.y - 5, 10, 10 );
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final MatrixMultiplyExample app = new MatrixMultiplyExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
}
```

## 3.7 仿射变换

Java 有一个名为 `AffineTransform` 的类，它可以用来执行矩阵变换。不用重新编写代码



是很好，但研究和学习算法以理解代码也是很好的。这个代码示例将继续从头开始开发库，但是，请放心地使用内建的类。

```
AffineTransform.setToIdentity()
AffineTransform.concatenate(AffineTransform)
AffineTransform.rotate(double radians)
AffineTransform.scale(double sx, double sy)
AffineTransform.translate(double tx, double ty)
AffineTransform.shear(double sx, double sy)
```

使用 **AffineTransform** 类，有很多方法将变换应用于数据点。如下的代码使用了所有可用的方法：

```
AffineTransform trans = new AffineTransform();
Shape srcShape = new Rectangle();
Shape destShape = null;
Point2D srcPoint = new Point2D.Float();
Point2D destPoint = new Point2D.Float();
Point2D pointSrc[] = new Point2D[] {
    new Point2D.Float(),
    new Point2D.Float(),
    new Point2D.Float()
};
Point2D pointDest[] = new Point2D[3];
double[] doubleSrc = new double[6];
double[] doubleDest = new double[6];
float[] floatSrc = new float[6];
float[] floatDest = new float[6];
int srcOff = 0;
int dstOff = 0;
int numPts = 3; // array is (x,y) pairs...
// Point -> Point
trans.transform( srcPoint, destPoint );
// Point[] -> Point[]
trans.transform( pointSrc, srcOff, pointDest, dstOff, numPts );
// double[] -> double[]
trans.transform( doubleSrc, srcOff, doubleDest, dstOff, numPts );
// float[] -> float[]
trans.transform( floatSrc, srcOff, floatDest, dstOff, numPts );
// double[] -> float[]
trans.transform( doubleSrc, srcOff, floatDest, dstOff, numPts );
// float[] -> double[]
```



```
trans.transform( floatSrc, srcOff, doubleDest, dstOff, numPts );
// Shape -> Shape
destShape = trans.createTransformedShape( srcShape );
```

这里给出的最后一个示例，将一个 Shape 变换成另外一个，这展示了其他操作（例如物理或冲突检测）获取变换的坐标的难度。可以使用一个 PathIterator 获取最终的数据，但是，它的用法有点难以理解。路径迭代可能返回数据的各种路径：

- SEG\_MOVETO——一条新的路径的起始位置；
- SEG LINETO——一条线段的第二个点；
- SEG\_QUADTO——确定一条二次参数曲线的两个点；
- SEG\_CUBICTO——确定一条三次参数曲线的三个点。

注意，可能返回 3 个(x,y)对。如下的代码变换一个矩形，并且遍历最终的变换：

```
AffineTransform trans = new AffineTransform();
trans.rotate( Math.PI / 3.0 );
trans.translate( 5.0, 7.0 );
Shape shape = new Rectangle( 0, 0, 100, 150 );
Shape transformed = trans.createTransformedShape( shape );
PathIterator pit = transformed.getPathIterator( null );
float[] seg = new float[6];
while( !pit.isDone() ) {
    int segType = pit.currentSegment( seg );
    String point = "(" + seg[0] + "," + seg[1] +
                   "), (" + seg[2] + "," + seg[3] +
                   "), (" + seg[4] + "," + seg[5] + ")";
    switch( segType ) {
        case PathIterator.SEG_MOVETO:
            System.out.println( "SEG_MOVETO: " + point );
            break;
        case PathIterator.SEG LINETO:
            System.out.println( "SEG LINETO: " + point );
            break;
        case PathIterator.SEG_QUADTO:
            System.out.println( "SEG QUADTO: " + point );
            break;
        case PathIterator.SEG_CUBICTO:
            System.out.println( "SEG_CUBICTO: " + point );
            break;
        case PathIterator.SEG_CLOSE:
            System.out.println( "SEG CLOSE: " + point );
            break;
    }
}
```



```
        break;  
    }  
    pit.next();  
}  
}
```

## 3.8 资源和延伸阅读

Akenine-Moller, Tomas and Eric Haines, *Real-Time Rendering*, 2<sup>nd</sup> Edition, A K Peters, 2002.

Ericson, Christer, *Real-Time Collision Detection*, Focal Press, 2004.

# 第4章

## 时间和空间

在我学习游戏编程时，我想要做一些涉及物理的事情；我查找了一些公式并且阅读有关运动的一些教程。然后，我编写了一些涉及重力的代码，并尝试正确地实现它们，以便使我的计算机程序中重力都是一样的，因为它就在地球之上。当我读到重力是  $9.8 \text{ m/s}^2$  时，我停了下来，盯着计算机好一会儿，不知道该从哪里下手。我没有任何像仪表或秒表一样的工具。我束手无策。就是在这个时候，我开始意识到计算机游戏有多么得难。本章的目标是给游戏循环添加时间步长，以考虑到那些需要时间的计算。本章还介绍了屏幕映射，以使得任意的坐标系统都可以绘制到任意大小的一个窗口上。

### 4.1 计算时间增量

之前的所有示例都是在每一帧上更新移动的物体。这导致不同的计算机根据帧速率的不同，以较快或较慢的速度来移动物体，如图 4.1 所示。

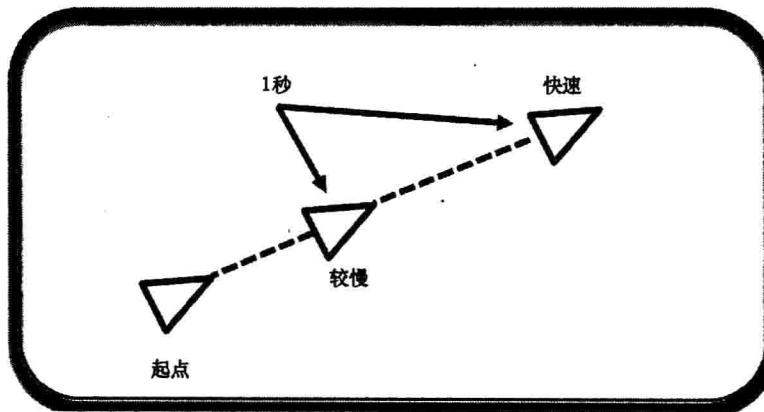


图 4.1 计时的移动



为了调整参数并避免游戏太容易或太难，物体需要以一致的速度移动，而不管计算机处理器的速度如何。有些时候，计算机会太旧并且太慢，但是，对任何其他的计算机来说，速度也都会不同。如果还有其他的应用程序在运行，计算机可能显得一天很快而第二天又很慢。

解决方案是计算帧之间的消耗时间，并且使用该值来缩放更新。

```
last_ns = last tick count  
curr_ns = current tick count  
elapsed_time_ns = curr_ns - last_ns  
elapsed_seconds = elapsed_time_ns / 1.0E9
```

注意，1秒等于 1.0E9 纳秒。用消耗时间除以每秒的纳秒数，得到每秒的消耗时间。最初，我的代码使用了毫秒，但是，我的处理器如此之快，以至于精度不够，并且没有物体移动。尽管它可能在较慢的计算机上能够工作，但还是应该坚持使用纳秒。

更新了 run()方法以计算消耗时间：

```
public void run() {  
    running = true;  
    initialize();  
    long curTime = System.nanoTime();  
    long lastTime = curTime;  
    double nsPerFrame;  
  
    while( running ) {  
        curTime = System.nanoTime();  
        nsPerFrame = curTime - lastTime;  
        gameLoop( nsPerFrame / 1.0E9 );  
        lastTime = curTime;  
    }  
}
```

一旦每帧的消耗时间可用，则可以使用它来缩放移动值，从而使得基于帧速率在屏幕上移动的距离不会改变：

$$\text{新的位置} = \text{旧的位置} + \text{帧速率} * \text{时间增量}$$

TimeDeltaExample 示例如图 4.2 所示，它位于 javagames.timeandspace 包中，它创建了一个旋转的计量器，很像是钟表。指针以每 4 秒一圈的速率旋转。箭头键用来增加或减小帧速率，以确保旋转的速度不会根据帧速率而变化。

首先需要注意的是，画布充当一个类变量，如果窗口大小变化的话，它也会调整钟表的大小。

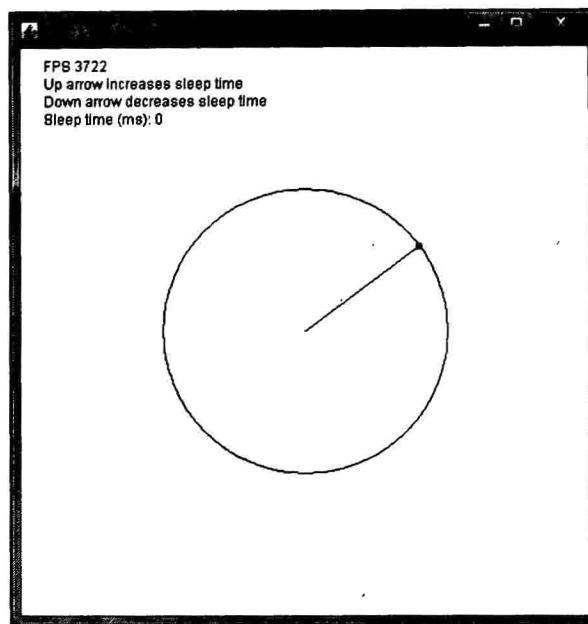


图 4.2 时间增量示例

其次需要注意的是，睡眠值也是一个变量，因此，在应用程序运行的时候，帧速率可以调整。当按下向上或向下箭头的时候，睡眠值调整 10 毫秒的幅度。这个值总是小于 1 秒钟，因为在轮询键盘的时候，如果帧速率降得过低，会开始漏掉键盘按下事件（我们将在第 11 章学习如何修正这个问题）。

`run()`方法已经更新了，它以秒为单位计算两帧之间的时间，并且将这个值传递给游戏循环。游戏循环现在把消耗时间传递给需要它的任何方法。例如，当处理输入的时候，没有任何理由需要消耗时间，但是随后会需要它。

`updateObjects()`方法是要使用新值的地方。加到 `angle` 中的 `step` 值，设置为  $\text{PI} / 2$  的弧度，或者说 90 度的角度。通过将其与消耗时间相乘，指针每秒钟移动 90 度。很容易看到，如果一秒钟过去，增加到 `angle` 的值将会是  $\text{PI} / 2$ 。如果消耗时间是半秒，那么增加的值就是 `angle` 的一半，或者说  $\text{PI} / 4$ 。

```
private void updateObjects( double delta ) {
    angle += step * delta;
    if( angle > 2 * Math.PI ) {
        angle -= 2 * Math.PI;
    }
}
```



render()方法打印出控件以及当前的睡眠时间。绘制圆圈的代码略有不同，当窗口大小调整时，它使用屏幕的宽度和高度来调整显示大小。

```
package javagames.timeandspace;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javagames.util.*;
import javax.swing.*;

public class TimeDeltaExample extends JFrame implements Runnable {
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private Canvas canvas;
    private float angle;
    private float step;
    private long sleep;
    public TimeDeltaExample() {
    }
    protected void createAndShowGUI() {
        canvas = new Canvas();
        canvas.setSize( 480, 480 );
        canvas.setBackground( Color.WHITE );
        canvas.setIgnoreRepaint( true );
        getContentPane().add( canvas );
        setTitle( "Time Delta Example" );
        setIgnoreRepaint( true );
        pack();
        keyboard = new KeyboardInput();
        canvas.addKeyListener( keyboard );
        mouse = new RelativeMouseInput( canvas );
        canvas.addMouseListener( mouse );
        canvas.addMouseMotionListener( mouse );
        canvas.addMouseWheelListener( mouse );
        setVisible( true );
        canvas.createBufferStrategy( 2 );
        bs = canvas.getBufferStrategy();
        canvas.requestFocus();
        gameThread = new Thread( this );
    }
}
```



```
        gameThread.start();
    }
    public void run() {
        running = true;
        initialize();
        long curTime = System.nanoTime();
        long lastTime = curTime;
        double nsPerFrame;
        while( running ) {
            curTime = System.nanoTime();
            nsPerFrame = curTime - lastTime;
            gameLoop( nsPerFrame / 1.0E9 );
            lastTime = curTime;
        }
    }
    private void gameLoop( double delta ) {
        processInput( delta );
        updateObjects( delta );
        renderFrame();
        sleep( sleep );
    }
    private void renderFrame() {
        do {
            do {
                Graphics g = null;
                try {
                    g = bs.getDrawGraphics();
                    g.clearRect( 0, 0, getWidth(), getHeight() );
                    render( g );
                } finally {
                    if( g != null ) {
                        g.dispose();
                    }
                }
            }
            } while( bs.contentsRestored() );
        bs.show();
    } while( bs.contentsLost() );
}
private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}
private void initialize() {
```



```
frameRate = new FrameRate();
frameRate.initialize();
angle = 0.0f;
step = (float)Math.PI / 2.0f;
}
private void processInput( double delta ) {
    keyboard.poll();
    mouse.poll();
    if( keyboard.keyDownOnce( KeyEvent.VK_UP ) ) {
        sleep += 10;
    }
    if( keyboard.keyDownOnce( KeyEvent.VK_DOWN ) ) {
        sleep -= 10;
    }
    if( sleep > 1000 ) {
        sleep = 1000;
    }
    if( sleep < 0 ) {
        sleep = 0;
    }
}
private void updateObjects( double delta ) {
    angle += step * delta;
    if( angle > 2 * Math.PI ) {
        angle -= 2 * Math.PI;
    }
}
private void render( Graphics g ) {
    g.setColor( Color.BLACK );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );
    g.drawString( "Up arrow increases sleep time", 20, 35 );
    g.drawString( "Down arrow decreases sleep time", 20, 50 );
    g.drawString( "Sleep time (ms): " + sleep, 20, 65 );
    int x = canvas.getWidth() / 4;
    int y = canvas.getHeight() / 4;
    int w = canvas.getWidth() / 2;
    int h = canvas.getHeight() / 2;
    g.drawOval( x, y, w, h );
    // polar -> coords
    float rw = w / 2; // radius width
    float rh = h / 2; // radius height
    int rx = (int)( rw * Math.cos( angle ) );
```



```
int ry = (int)( rh * Math.sin( angle ) );
int cx = (int)(rx + w);
int cy = (int)(ry + h);
// draw clock hand
g.drawLine( w, h, cx, cy );
// draw dot at end of hand
g.drawRect( cx - 2, cy - 2, 4, 4 );
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final TimeDeltaExample app = new TimeDeltaExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
```

## 4.2 屏幕映射

当你想要使用物理方程来旋转移动对象的时候，需要具备两个条件，而这在前面的例子中是不满足的。以 10 m/s 移动，意味着物体每秒钟要移动 10 米。秒已经添加了，那么，米呢？

在前面的示例中，所使用的一些值并不直观，并且其行为并不像预期的那样。这是因

为屏幕坐标并不和学校里所教授的笛卡尔坐标系统匹配，如图 4.3 所示。

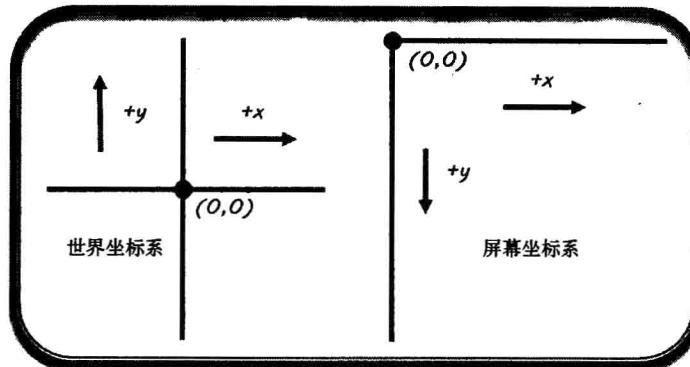


图 4.3 世界坐标系和屏幕坐标系映射

如果我们能够在任意的坐标系统中定义物体，然后将它们映射到屏幕，那就好了。事实上，要做到这一点很简单，使用第 3 章介绍的变换就可以了。创建行为如预期一样的坐标系统，在屏幕上缩放映射的值以使得窗口的大小变得无关紧要，这两点都很重要。游戏在不同的设备上具有不同大小的窗口，这是很重要的，如图 4.4 所示。

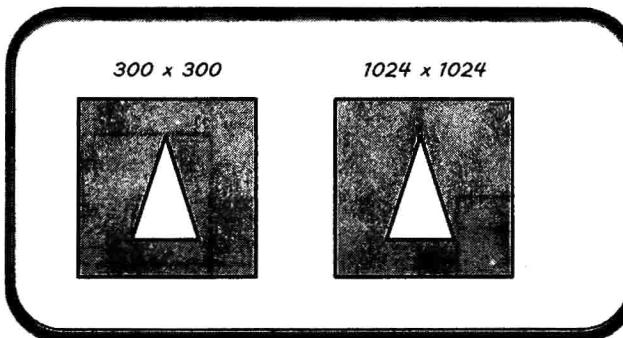


图 4.4 多种大小

较大的窗口可以缩放所有内容，保持物体具有相同的比例，而不管窗口占据多大的大小。起初，我们必须定义一个宽度和高度，它们定义了游戏的世界。很多图形化系统使用一个 (-1,1) 窗口，如图 4.5 所示。

这会定义一个  $2 \times 2$  单位的区域并映射到屏幕上，你可以使用想要的任何单位。如果游戏需要米，就称之为米。如果游戏是一个行星游戏，那么它应该是英里或者光年。这无关紧要，但是它可以帮助确定物体的大小，并且能够在屏幕上产生一种距离感。例如，假设单位为米，可用的区域是  $2 \times 2$  米或者  $6.56 \times 6.56$  英尺。

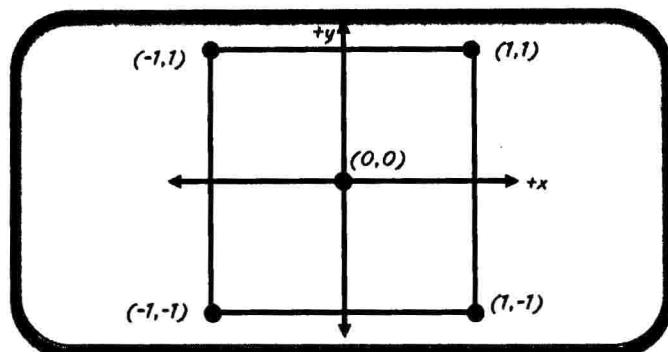


图 4.5 常规化的设备坐标

要把世界区域映射到屏幕上，首先要将该区域缩放到屏幕的宽度和高度。接下来，翻转 y 轴。最后，平移该区域，以便当这些值绘制到屏幕上的时候是正确的，如图 4.6 所示。

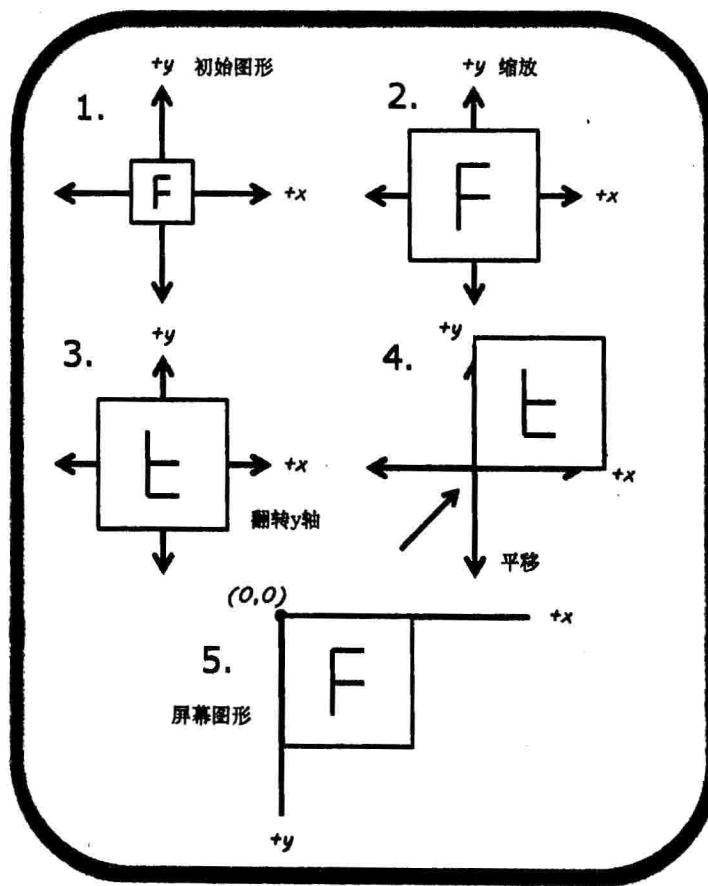


图 4.6 屏幕映射算法

如下的代码用来计算将世界坐标系转换为屏幕坐标系的视口矩阵。注意，在这个例子中，缩放和平移值是 `width / 2` 且 `height / 2`。这是因为(-1,1)世界坐标的宽度和高度都是 2。例如，如果世界区域是 5x5，那么，缩放值应该除以 5，但是平移值总是除以 2，并且世界坐标系的中心位于(0,0)。

```
float worldWidth = 2.0f;
float worldHeight = 2.0f;
float screenWidth = canvas.getWidth() - 1;
float screenHeight = canvas.getHeight() - 1;
float sx = screenWidth / worldWidth;
float sy = screenHeight / worldHeight;
float tx = screenWidth / 2.0f;
float ty = screenHeight / 2.0f;
Matrix3x3f viewport = Matrix3x3f.scale( sx, -sy );
viewport = viewport.mul( Matrix3x3f.translate( tx, ty ) );
```

注意，在前面的示例代码中，在使用屏幕的宽度和高度之前，先将它们减去 1。这是因为实际的像素值映射到了屏幕上，并且尽管屏幕的宽度可能是 10，但像素从 0 到 9，就像是一个数组一样。如果使用实际的宽度，那么当世界坐标系转换为屏幕坐标系的时候，最后一行和最后一列将不会绘制到屏幕上，而这不符合本来的意图。将屏幕的宽度设置为 `width - 1`，并且将屏幕的高度设置为 `height - 1`，就可以避免发生这种情况。

`ScreenMappingExample` 如图 4.7 所示，位于 `javagames.timeandspace` 包中，这是一个简单的程序，它允许屏幕调整大小并同时保证宽高比与窗口的宽度和高度相匹配。

有一个三角数组及一个名为 `triWorld` 的副本，用来保存映射的值。在 `initialize()` 方法中，这个三角数组的值现在还很小。因为世界的大小为  $2 \times 2$ ，所以用非常小的值生成一个三角来填充屏幕。

`render()` 方法创建了视口矩阵并与三角的点相乘，将世界空间中的值映射到屏幕空间。此外，围绕边界绘制了一个边框矩形。尝试从屏幕宽度和屏幕高度的计算中去除掉-1，并且看看边框矩形发生了什么变化。

```
package javagames.timeandspace;

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

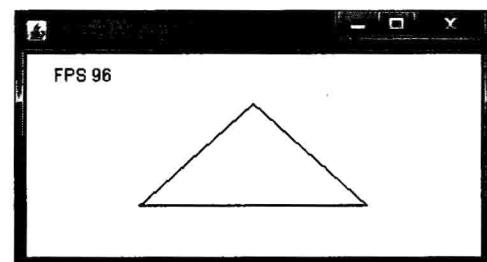


图 4.7 屏幕映射示例



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

public class ScreenMappingExample extends JFrame
    implements Runnable {
    private Canvas canvas;
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private Vector2f[] tri;
    private Vector2f[] triWorld;
    private Vector2f[] rect;
    private Vector2f[] rectWorld;
    public ScreenMappingExample() {
    }
    protected void createAndShowGUI() {
        canvas = new Canvas();
        canvas.setSize( 640, 480 );
        canvas.setBackground( Color.WHITE );
        canvas.setIgnoreRepaint( true );
        getContentPane().add( canvas );
        setTitle( "Screen Mapping Example" );
        setIgnoreRepaint( true );
        pack();
        keyboard = new KeyboardInput();
        canvas.addKeyListener( keyboard );
        mouse = new RelativeMouseInput( canvas );
        canvas.addMouseListener( mouse );
        canvas.addMouseMotionListener( mouse );
        canvas.addMouseWheelListener( mouse );
        setVisible( true );
        canvas.createBufferStrategy( 2 );
        bs = canvas.getBufferStrategy();
        canvas.requestFocus();
        gameThread = new Thread( this );
        gameThread.start();
    }
    public void run() {
        running = true;
```



```
initialize();
long curTime = System.nanoTime();
long lastTime = curTime;
double nsPerFrame;
while( running ) {
    curTime = System.nanoTime();
    nsPerFrame = curTime - lastTime;
    gameLoop( nsPerFrame / 1.0E9 );
    lastTime = curTime;
}
private void initialize() {
    frameRate = new FrameRate();
    frameRate.initialize();
    tri = new Vector2f[] {
        new Vector2f( 0.0f, 0.5f ),
        new Vector2f( -0.5f, -0.5f ),
        new Vector2f( 0.5f, -0.5f ),
    };
    triWorld = new Vector2f[ tri.length ];
    rect = new Vector2f[] {
        new Vector2f( -1.0f, 1.0f ),
        new Vector2f( 1.0f, 1.0f ),
        new Vector2f( 1.0f, -1.0f ),
        new Vector2f( -1.0f, -1.0f ),
    };
    rectWorld = new Vector2f[ rect.length ];
}
private void gameLoop( double delta ) {
    processInput( delta );
    updateObjects( delta );
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
            try {
                g = bs.getDrawGraphics();
                g.clearRect( 0, 0, getWidth(), getHeight() );
                render( g );
            } finally {

```



```
        if( g != null ) {
            g.dispose();
        }
    }
    while( bs.contentsRestored() );
    bs.show();
} while( bs.contentsLost() );
}

private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}

private void processInput( double delta ) {
    keyboard.poll();
    mouse.poll();
}

private void updateObjects( double delta ) {
}

private void render( Graphics g ) {
    g.setColor( Color.BLACK );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );
    float worldWidth = 2.0f;
    float worldHeight = 2.0f;
    float screenWidth = canvas.getWidth() - 1;
    float screenHeight = canvas.getHeight() - 1;
    float sx = screenWidth / worldWidth;
    float sy = screenHeight / worldHeight;
    float tx = screenWidth / 2.0f;
    float ty = screenHeight / 2.0f;
    Matrix3x3f viewport = Matrix3x3f.scale( sx, -sy );
    viewport = viewport.mul( Matrix3x3f.translate( tx, ty ) );
    for( int i = 0; i < tri.length; ++i ) {
        triWorld[i] = viewport.mul( tri[i] );
    }
    drawPolygon( g, triWorld );
    for( int i = 0; i < rect.length; ++i ) {
        rectWorld[i] = viewport.mul( rect[i] );
    }
    drawPolygon( g, rectWorld );
}

private void drawPolygon( Graphics g, Vector2f[] polygon ) {
```



```
Vector2f P;
Vector2f S = polygon[ polygon.length - 1 ];
for( int i = 0; i < polygon.length; ++i ) {
    P = polygon[i];
    g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );
    S = P;
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final ScreenMappingExample app = new ScreenMappingExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
```

## 4.3 调整视口高宽比

在上一个示例中，三角形展开并填充了窗口，而不管其大小是多少。调整窗口的大小而将三角形变得与其最初大小相去甚远，这也是可能发生的，但这可能不是窗口游戏应该有的行为。尽管能够映射到不同大小的视口是很重要的，但保持同样的高宽比也很重要，这样一来，不管窗口多大或者多小，都能保持正确的比例。

当你添加一个 `ComponentListener` 的时候，任何时候，只要窗口改变大小，它都会通知应用程序。因此，不管窗口有多大，它的高宽比始终是相同的。`ComponentListener` 有几个接口，但是，要检测改变大小事件，只需要 `componentResized()` 方法：

```
componentHidden( ComponentEvent e )
componentMoved( ComponentEvent e )
componentResized( ComponentEvent e )
componentShown( ComponentEvent e )
```

一旦检测到改变大小事件，就需要调整视口以保持相同的高宽比。根据窗口的大小和视口的大小，可能会出现一些有趣的尺寸，如图 4.8 所示。

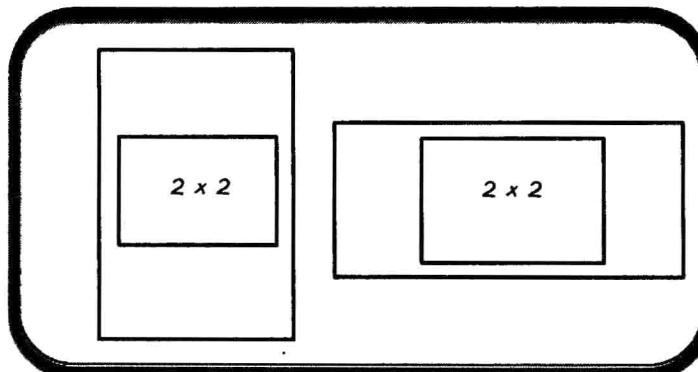


图 4.8 高宽比

如果调整后的视口在当前窗口中居中，那么，不管窗口多大，视口都将增大或缩小到可能的最大尺寸，并且仍然保持正确的高宽比。

使用两个窗口的高宽比，来得到新的尺寸：

$$\frac{\text{new width}}{\text{new height}} = \frac{\text{viewport width}}{\text{viewport height}}$$
$$\text{new width} = \text{new height} * \frac{\text{viewport width}}{\text{viewport height}}$$
$$\text{new height} = \text{new width} * \frac{\text{viewport height}}{\text{viewport width}}$$

要计算视口的新的尺寸，可以选择屏幕宽度或屏幕高度来计算另外一个值，从而保持正确的高宽比。尽管你可以执行检查来确定使用哪个尺寸，但任意选取一个尺寸并检查新的高宽比是否符合当前窗口是很方便的，如果使用一个值所得到的高宽比太大的话，就使用另一个值。



```

int newW = vw;
int newH = (int)(vw * worldHeight / worldWidth);
if( newH > vh ) {
    newW = (int)(vh * worldWidth / worldHeight);
    newH = vh;
}

```

如图 4.9 所示, ViewportRatioExample 位于 javagames.timeandspace 包中, 不管屏幕大小是多少, 它保持世界高宽比的大小。首先要注意的是 ComponentAdapter, 它添加到了内容面板上。也可以给 JFrame 添加该监听器, 但是, 只有在鼠标按键释放时, 它才会接收到改变大小的事件。根据游戏的行为, 这么做可能是必要的, 而不是当窗口改变大小时重新绘制它。

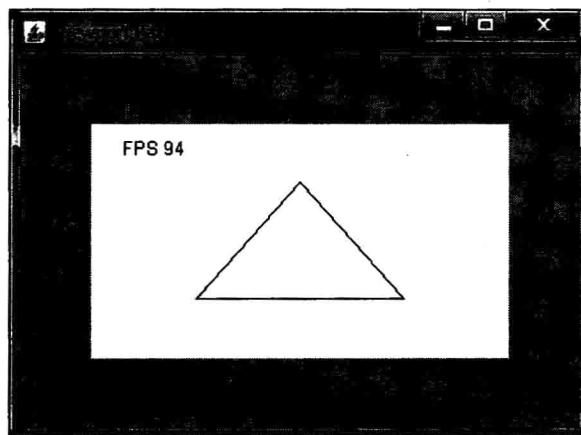


图 4.9 视口高宽比示例

```

getContentPane().addComponentListener( new ComponentAdapter() {
    public void componentResized( ComponentEvent e ) {
        onComponentResized( e );
    }
});

```

世界的高宽比是 16:9, 这是 initialize()方法中的一个宽屏的世界。更改这些值, 将会修改视口的高宽比。

```

worldWidth = 16.0f;
worldHeight = 9.0f;

```

真正改变大小, 是通过计算画布对象的一个新的(x,y)位置和一个新的宽度和高度而完成的, 如图 4.10 所示。

改变大小的过程如下。



1. 初始图形。在 componentResized()事件内部，新的尺寸变得可用。
2. 缩小。调整当前的尺寸。在这个例子中，它改变为当前尺寸的 $\frac{3}{4}$ 。通过比较初始尺寸和 $\frac{3}{4}$ 尺寸的区别，并且剪裁掉剩余区域的一半，从而计算出中心点。
3. 使用调整后较小的尺寸来计算新的宽度和高度，这里将会有一个边框。
4. 调整后的尺寸在较小的尺寸中居中，就像前面的视口居中一样。

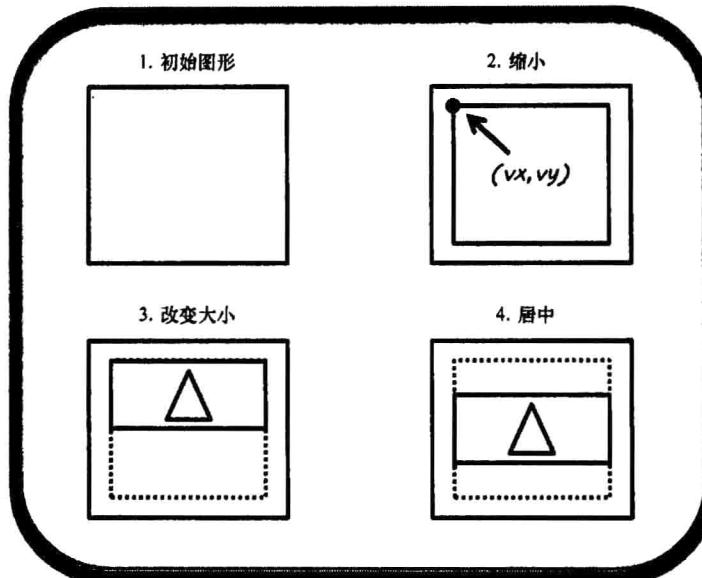


图 4.10 改变大小和居中的算法

该示例的代码如下：

```
package javagames.timeandspace;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javagames.util.*;
import javax.swing.*;

public class ViewportRatio extends JFrame implements Runnable {
    private FrameRate frameRate;
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    private RelativeMouseInput mouse;
    private KeyboardInput keyboard;
    private Canvas canvas;
```



```
private Vector2f[] tri;
private Vector2f[] triWorld;
private float worldWidth;
private float worldHeight;
public ViewportRatio() {
}
protected void createAndShowGUI() {
    canvas = new Canvas();
    canvas.setBackground( Color.WHITE );
    canvas.setIgnoreRepaint( true );
    getContentPane().setBackground( Color.LIGHT_GRAY );
    setLayout( null );
    setTitle( "Viewport Ratio" );
    setSize( 640, 640 );
    getContentPane().add( canvas );
    keyboard = new KeyboardInput();
    canvas.addKeyListener( keyboard );
    mouse = new RelativeMouseInput( canvas );
    canvas.addMouseListener( mouse );
    canvas.addMouseMotionListener( mouse );
    canvas.addMouseWheelListener( mouse );
    getContentPane().addComponentListener( new ComponentAdapter() {
        public void componentResized( ComponentEvent e ) {
            onComponentResized( e );
        }
    });
    setVisible( true );
    canvas.createBufferStrategy( 2 );
    bs = canvas.getBufferStrategy();
    canvas.requestFocus();
    gameThread = new Thread( this );
    gameThread.start();
}
protected void onComponentResized( ComponentEvent e ) {
    Dimension size = getContentPane().getSize();
    int vw = size.width * 3 / 4;
    int vh = size.height * 3 / 4;
    int vx = (size.width - vw) / 2;
    int vy = (size.height - vh) / 2;
    int newW = vw;
    int newH = (int)(vw * worldHeight / worldWidth);
    if( newH > vh ) {
        newW = (int)(vh * worldWidth / worldHeight);
```



```
    newH = vh;
}
// center
vx += (vw - newW) / 2;
vy += (vh - newH) / 2;
canvas.setLocation( vx, vy );
canvas.setSize( newW, newH );
}
public void run() {
    running = true;
    initialize();
    long curTime = System.nanoTime();
    long lastTime = curTime;
    double nsPerFrame;
    while( running ) {
        curTime = System.nanoTime();
        nsPerFrame = curTime - lastTime;
        gameLoop( nsPerFrame / 1.0E9 );
        lastTime = curTime;
    }
}
private void gameLoop( double delta ) {
    processInput( delta );
    updateObjects( delta );
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
            try {
                g = bs.getDrawGraphics();
                g.clearRect( 0, 0, getWidth(), getHeight() );
                render( g );
            } finally {
                if( g != null ) {
                    g.dispose();
                }
            }
        }
    } while( bs.contentsRestored() );
    bs.show();
} while( bs.contentsLost() );
```



```
}

private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}

private void initialize() {
    frameRate = new FrameRate();
    frameRate.initialize();
    tri = new Vector2f[] {
        new Vector2f( 0.0f, 2.25f ),
        new Vector2f(-4.0f,-2.25f ),
        new Vector2f( 4.0f,-2.25f ),
    };
    triWorld = new Vector2f[ tri.length ];
    worldWidth = 16.0f;
    worldHeight = 9.0f;
}
private void processInput( double delta ) {
    keyboard.poll();
    mouse.poll();
}
private void updateObjects( double delta ) {
}
private void render( Graphics g ) {
    g.setColor( Color.BLACK );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );

    float sx = (canvas.getWidth() - 1) / worldWidth;
    float sy = (canvas.getHeight() - 1) / worldHeight;
    float tx = (canvas.getWidth() - 1) / 2.0f;
    float ty = (canvas.getHeight() - 1) / 2.0f;
    Matrix3x3f viewport = Matrix3x3f.identity();
    viewport = viewport.mul( Matrix3x3f.scale( sx, -sy ) );
    viewport = viewport.mul( Matrix3x3f.translate( tx, ty ) );
    for( int i = 0; i < tri.length; ++i ) {
        triWorld[i] = viewport.mul( tri[i] );
    }
    drawPolygon( g, triWorld );
}
private void drawPolygon( Graphics g, Vector2f[] polygon ) {
    Vector2f P;
```



```
Vector2f S = polygon[ polygon.length - 1 ];
for( int i = 0; i < polygon.length; ++i ) {
    P = polygon[i];
    g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );
    S = P;
}
protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
public static void main( String[] args ) {
    final ViewportRatio app = new ViewportRatio();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
```

## 4.4 大炮实例

CannonExample 如图 4.11 所示，位于 `javagames.timeandspace` 包中，它向空中射出一个小方块，就像是一颗大炮炮弹一样。重要的是，理解在移动物体的时候如何使用时间增量参数。

确定一个物体位置的标准物理公式如下：

$$d = d_0 + vt - \frac{at^2}{2}$$



其中( $d_0$ )是初始位置, ( $v$ )是速率, ( $a$ )是加速度, ( $t$ )是时间, ( $d$ )是最终位置。使用这个公式的问题在于, 它需要完整的时间, 而不只是两帧之间的时间。对于短暂的时间, 如下的公式更为有用:

$$v = \frac{d}{t}, a = \frac{v}{t}, d = vt, v = at$$

这些公式只使用当前时间, 而不是完整的时间。务必要注意需要什么样的时间值。如果需要完整的时间, 那么, 不要只是使用消耗时间。

`initialize()`方法设置了默认值, 创建了大炮模型, 并且将其缩放 0.75。当按下 A 或 D 键时, 会旋转大炮, 并且使用时间增量来控制旋转。在前面的示例中, 旋转增量设置为一个较小的值, 如 5 度。现在, 将其设置为每秒 90 度, 并且使用时间增量来递增。

当用户按下空格键时, 会创建一个新的炮弹。通过在 x 轴上创建一个 7 单位的向量, 然后将其旋转到正确的角度, 从而计算速率。也可以使用极坐标来计算速率, 但是, 我想要展示使用矩阵的另一种方法。

将炮弹变换到旋转的大炮的末端。之所以使用值 0.375, 是因为在使用大炮模型之前, 最初的大炮长度 0.5 缩放了 0.75。

`update()`函数定位并复制大炮模型, 以使其准备好渲染。如果有炮弹, 调整其位置。我们没有使用向量来表示重力, 而是使用一个 -9.8 的值来替代。现在视口映射已经准备好了, 正的 y 值在屏幕上向下移动, 我可以使用 -9.8 作为重力参数。如果没有映射视口的话, y 的单位将会翻转, 这时需要对重力使用一个正值才能让炮弹落地, 这就有点怪异了。

渲染方法显示了通常的帧速率和说明。这个示例的世界空间是 5x5 米, 这会映射到屏幕上。这里没有保持高宽比, 因此, 调整屏幕大小将会导致一些奇怪的事情。在应用了视口转换之后, 将会绘制大炮和炮弹。

```
package javagames.timeandspace;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javagames.util.*;
import javax.swing.*;

public class CannonExample extends JFrame
```

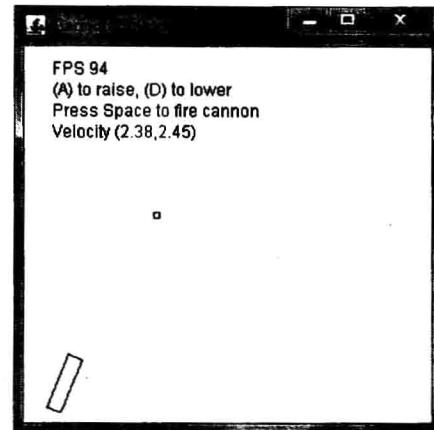


图 4.11 大炮实例



```
    implements Runnable {  
  
        private FrameRate frameRate;  
        private BufferStrategy bs;  
        private volatile boolean running;  
        private Thread gameThread;  
        private RelativeMouseInput mouse;  
        private KeyboardInput keyboard;  
        private Canvas canvas;  
  
        private Vector2f[] cannon;  
        private Vector2f[] cannonCpy;  
        private float cannonRot, cannonDelta;  
  
        private Vector2f bullet;  
        private Vector2f bulletCpy;  
        private Vector2f velocity;  
  
        public CannonExample() {  
            }  
  
        protected void createAndShowGUI() {  
  
            canvas = new Canvas();  
            canvas.setSize( 640, 480 );  
            canvas.setBackground( Color.WHITE );  
            canvas.setIgnoreRepaint( true );  
            getContentPane().add( canvas );  
            setTitle( "Cannon Example" );  
            setIgnoreRepaint( true );  
            pack();  
  
            keyboard = new KeyboardInput();  
            canvas.addKeyListener( keyboard );  
            mouse = new RelativeMouseInput( canvas );  
            canvas.addMouseListener( mouse );  
            canvas.addMouseMotionListener( mouse );  
            canvas.addMouseWheelListener( mouse );  
  
            setVisible( true );  
            canvas.createBufferStrategy( 2 );  
            bs = canvas.getBufferStrategy();
```



```
canvas.requestFocus();

gameThread = new Thread( this );
gameThread.start();
}

public void run() {
    running = true;
    initialize();
    long curTime = System.nanoTime();
    long lastTime = curTime;
    double nsPerFrame;
    while( running ) {
        curTime = System.nanoTime();
        nsPerFrame = curTime - lastTime;
        gameLoop( nsPerFrame / 1.0E9 );
        lastTime = curTime;
    }
}

private void initialize() {

    frameRate = new FrameRate();
    frameRate.initialize();

    velocity = new Vector2f();
    cannonRot = 0.0f;
    cannonDelta = (float)Math.toRadians( 90.0 );
    cannon = new Vector2f[] {
        new Vector2f( -0.5f, 0.125f ), // top-left
        new Vector2f( 0.5f, 0.125f ), // top-right
        new Vector2f( 0.5f, -0.125f ), // bottom-right
        new Vector2f( -0.5f, -0.125f ), // bottom-left
    };
    cannonCpy = new Vector2f[ cannon.length ];

    Matrix3x3f scale = Matrix3x3f.scale( .75f, .75f );
    for( int i = 0; i < cannon.length; ++i ) {
        cannon[i] = scale.mul( cannon[i] );
    }
}

private void gameLoop( double delta ) {
```



```
    processInput( delta );
    updateObjects( delta );
    renderFrame();
    sleep( 10L );
}
private void renderFrame() {
    do {
        do {
            Graphics g = null;
            try {
                g = bs.getDrawGraphics();
                g.clearRect( 0, 0, getWidth(), getHeight() );
                render( g );
            } finally {
                if( g != null ) {
                    g.dispose();
                }
            }
        }
        while( bs.contentsRestored() );
        bs.show();
    } while( bs.contentsLost() );
}

private void sleep( long sleep ) {
    try {
        Thread.sleep( sleep );
    } catch( InterruptedException ex ) { }
}

private void processInput( double delta ) {

    keyboard.poll();
    mouse.poll();

    if( keyboard_KeyDown( KeyEvent.VK_A ) ) {
        cannonRot += cannonDelta * delta;
    }
    if( keyboard_KeyDown( KeyEvent.VK_D ) ) {
        cannonRot -= cannonDelta * delta;
    }
    if( keyboard_KeyDownOnce( KeyEvent.VK_SPACE ) ) {
        // new velocity
    }
}
```



```
Matrix3x3f mat = Matrix3x3f.translate( 7.0f, 0.0f );
mat = mat.mul( Matrix3x3f.rotate( cannonRot ) );
velocity = mat.mul( new Vector2f() );
// place bullet at cannon end
mat = Matrix3x3f.translate( .375f, 0.0f );
mat = mat.mul( Matrix3x3f.rotate( cannonRot ) );
mat = mat.mul( Matrix3x3f.translate( -2.0f, -2.0f ) );
bullet = mat.mul( new Vector2f() );
}

private void updateObjects( double delta ) {

    Matrix3x3f mat = Matrix3x3f.identity();
    mat = mat.mul( Matrix3x3f.rotate( cannonRot ) );
    mat = mat.mul( Matrix3x3f.translate( -2.0f, -2.0f ) );

    for( int i = 0; i < cannon.length; ++i ) {
        cannonCpy[i] = mat.mul( cannon[i] );
    }

    if( bullet != null ) {

        velocity.y += -9.8f * delta;
        bullet.x += velocity.x * delta;
        bullet.y += velocity.y * delta;
        bulletCpy = new Vector2f( bullet );

        if( bullet.y < -2.5f ) {
            bullet = null;
        }
    }
}

private void render( Graphics g ) {

    g.setColor( Color.BLACK );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );
    g.drawString( "(A) to raise, (D) to lower", 20, 35 );
    g.drawString( "Press Space to fire cannon", 20, 50 );
    String vel =
        String.format( "Velocity (%.2f,%.2f)", velocity.x, velocity.y );
}
```



```
g.drawString( vel, 20, 65 );

float worldWidth = 5.0f;
float worldHeight = 5.0f;
float screenWidth = canvas.getWidth() - 1;
float screenHeight = canvas.getHeight() - 1;
float sx = screenWidth / worldWidth;
float sy = -screenHeight / worldHeight;
Matrix3x3f viewport = Matrix3x3f.scale( sx, sy );
float tx = screenWidth / 2.0f;
float ty = screenHeight / 2.0f;
viewport = viewport.mul( Matrix3x3f.translate( tx, ty ) );

for( int i = 0; i < cannon.length; ++i ) {
    cannonCpy[i] = viewport.mul( cannonCpy[i] );
}

drawPolygon( g, cannonCpy );

if( bullet != null ) {
    bulletCpy = viewport.mul( bulletCpy );
    g.drawRect( (int)bulletCpy.x-2, (int)bulletCpy.y-2, 4, 4 );
}
}

private void drawPolygon( Graphics g, Vector2f[] polygon ) {
    Vector2f P;
    Vector2f S = polygon[ polygon.length - 1 ];
    for( int i = 0; i < polygon.length; ++i ) {
        P = polygon[i];
        g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );
        S = P;
    }
}

protected void onWindowClosing() {
    try {
        running = false;
        gameThread.join();
    } catch( InterruptedException e ) {
        e.printStackTrace();
    }
    System.exit( 0 );
}
```



```
}

public static void main( String[] args ) {
    final CannonExample app = new CannonExample();
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
            app.createAndShowGUI();
        }
    });
}
```

## 4.5 资源和延伸阅读

Sanglard, Fabien, “Game Timers: Issues and Solutions,” [http://fabiensanglard.net/timer\\_and\\_framerate/index.php](http://fabiensanglard.net/timer_and_framerate/index.php).

Fiedler, Glenn, “Fix Your Timestep,” <http://gafferongames.com/game-physics/fixyour-timestep/>.



# 第5章

## 简单游戏框架

到目前为止，大多示例都拥有很多相同的代码。前面所有的各章，都以某种方式为主游戏循环添加和更新了功能。由于每个示例的框架没有太多的更新，创建一个可以用作所有如下示例起点的 SimpleFramework 是有帮助的。

### 5.1 屏幕到世界的转换

Utility 类已经添加到了 `javagames.util` 包中，它拥有如下的一些方法用于创建视口转换矩阵：

```
package javagames.util;
public class Utility {
    public static Matrix3x3f createViewport(
        float worldWidth, float worldHeight,
        float screenWidth, float screenHeight ) {
        float sx = (screenWidth - 1) / worldWidth;
        float sy = (screenHeight - 1) / worldHeight;
        float tx = (screenWidth - 1) / 2.0f;
        float ty = (screenHeight - 1) / 2.0f;
        Matrix3x3f viewport = Matrix3x3f.scale( sx, -sy );
        viewport = viewport.mul( Matrix3x3f.translate( tx, ty ) );
        return viewport;
    }
    public static Matrix3x3f createReverseViewport(
        float worldWidth, float worldHeight,
        float screenWidth, float screenHeight ) {
        float sx = worldWidth / (screenWidth - 1);
        float sy = worldHeight / (screenHeight - 1);
```



```

        float tx = (screenWidth - 1) / 2.0f;
        float ty = (screenHeight - 1) / 2.0f;
        Matrix3x3f viewport = Matrix3x3f.translate( -tx, -ty );
        viewport = viewport.mul( Matrix3x3f.scale( sx, -sy ) );
        return viewport;
    }

}

```

注意 `createReverseViewport()` 是平移并缩放，而不是缩放并平移，它执行的是视口变换的相反的操作。这可以用来将鼠标屏幕坐标转换为世界坐标。

`getViewportTransform()` 和 `getReverseViewportTransform()` 已经作为工具方法添加了，使用画布大小和世界大小来创建变换矩阵。`getWorldMousePosition()` 用来把屏幕坐标转换为世界坐标。当要设置鼠标以返回相对位置时，要使用 `getRelativeWorldMousePosition()` 方法。

```

// SimpleFramework.java
protected Matrix3x3f getViewportTransform() {
    return Utility.createViewport(
        appWorldWidth, appWorldHeight, canvas.getWidth(), canvas.getHeight()
    );
}

protected Matrix3x3f getReverseViewportTransform() {
    return Utility.createReverseViewport(
        appWorldWidth, appWorldHeight, canvas.getWidth(), canvas.getHeight()
    );
}

protected Vector2f getWorldMousePosition() {
    Matrix3x3f screenToWorld = getReverseViewportTransform();
    Point mousePos = mouse.getPosition();
    Vector2f screenPos = new Vector2f( mousePos.x, mousePos.y );
    return screenToWorld.mul( screenPos );
}

protected Vector2f getRelativeWorldMousePosition() {
    float sx = appWorldWidth / (canvas.getWidth() - 1);
    float sy = appWorldHeight / (canvas.getHeight() - 1);
    Matrix3x3f viewport = Matrix3x3f.scale( sx, -sy );
    Point p = mouse.getPosition();
    return viewport.mul( new Vector2f( p.x, p.y ) );
}

```



## 5.2 理解简单框架

关于 `javagames.util` 包中的 `SimpleFramework` 类，首先需要注意的是其所有的属性。尽管有方法能够访问单个的属性，但很容易在构造方法中只是使用默认值或修改它们。由于所有的属性都是以 `app` 开头的，因此使用代码编辑器中的自动补齐功能将会看到可用属性的一个列表，如图 5.1 所示。

- `appBackground`——主游戏区的背景颜色。
- `appBorder`——保持高宽比时的边框颜色。
- `appFPSColor`——绘制每秒帧显示的颜色。
- `appFont`——显示每秒帧时使用的字体。
- `appTitle`——示例的标题。
- `appBorderStyle`——保持高宽比时，游戏区和边框区的比例。
- `appWidth`——窗口的初始宽度。
- `appHeight`——窗口的初始高度。
- `appWorldWidth`——世界坐标的宽度。
- `appWorldHeight`——世界坐标的高度。
- `appSleep`——每帧睡眠的毫秒数。
- `appMaintainRatio`——如果应该保持高宽比的话，为真。

这个类没有 `main()` 方法。子类可以使用 `launchApp()` 方法来添加关闭窗口监听器并启动应用程序。

```
// SimpleFramework.java
protected static void launchApp( final SimpleFramework app ) {
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {
            app.onWindowClosing();
        }
    });
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
```

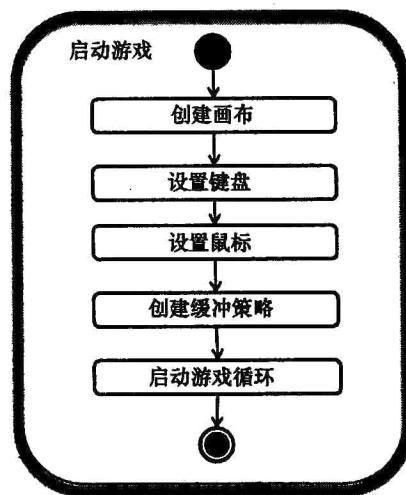


图 5.1 启动应用程序

```
        app.createAndShowGUI();  
    }  
});  
}
```

在 `createAndShowGUI()` 方法中，新的代码检查了高宽比属性，并且要么初始化应用程序以延展视图，要么在窗口改变大小时保持高宽比。参见第 4 章中的 `ViewportRatioExample` 方法以了解更多信息。当窗口改变大小并且应该保持高宽比时，使用 `onComponentResized()` 方法来调整画布大小，如图 5.2 所示。

`run()` 方法中添加了两个方法，以使得子类能够在主游戏循环之前和之后运行代码。

- `initialize()` 方法在游戏循环开始之前调用。
- `terminate()` 方法在应用程序关闭之后调用。

`initialize()` 方法创建并初始化了 `frameRate` 对象，`terminate()` 方法为空，并且游戏循环也是一样为空。在 `terminate()` 方法中没有执行什么，并且游戏循环也是如此。这 3 个用于执行游戏操作的方法声明为 `protected` 的，以便子类可以覆盖它们。

```
protected void processInput(float delta)  
protected void updateObjects(float delta)  
protected void render(Graphics g)
```

`processInput()` 方法针对键盘和鼠标调用轮询方法。`render()` 方法绘制帧速率，并且 `updateObjects()` 暂时是空的，如图 5.3 所示。



图 5.2 游戏线程

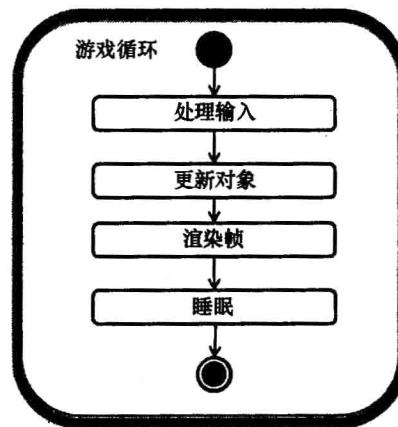


图 5.3 游戏循环

```
package javagames.util;  
import java.awt.*;
```



章 游戏框架

```
import java.awt.event.*;
import java.awt.image.BufferStrategy;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
public class SimpleFramework extends JFrame implements Runnable {
    private BufferStrategy bs;
    private volatile boolean running;
    private Thread gameThread;
    protected FrameRate frameRate;
    protected Canvas canvas;
    protected RelativeMouseInput mouse;
    protected KeyboardInput keyboard;
    protected Color appBackground = Color.BLACK;
    protected Color appBorder = Color.LIGHT_GRAY;
    protected Color appFPSColor = Color.GREEN;
    protected Font appFont = new Font( "Courier New", Font.PLAIN, 14 );
    protected String appTitle = "TBD-Title";
    protected float appBorderScale = 0.8f;
    protected int appWidth = 640;
    protected int appHeight = 480;
    protected float appWorldWidth = 2.0f;
    protected float appWorldHeight = 2.0f;
    protected long appSleep = 10L;
    protected boolean appMaintainRatio = false;
    public SimpleFramework() {
    }
    protected void createAndShowGUI() {
        canvas = new Canvas();
        canvas.setBackground( appBackground );
        canvas.setIgnoreRepaint( true );
        getContentPane().add( canvas );
        setLocationByPlatform( true );
        if( appMaintainRatio ) {
            getContentPane().setBackground( appBorder );
            setSize( appWidth, appHeight );
            setLayout( null );
            getContentPane().addComponentListener( new ComponentAdapter() {
                public void componentResized( ComponentEvent e ) {
                    onComponentResized( e );
                }
            });
        } else {
            canvas.setSize( appWidth, appHeight );
        }
    }
}
```



```
        pack();
    }
    setTitle( appTitle );
    keyboard = new KeyboardInput();
    canvas.addKeyListener( keyboard );
    mouse = new RelativeMouseInput( canvas );
    canvas.addMouseListener( mouse );
    canvas.addMouseMotionListener( mouse );
    canvas.addMouseWheelListener( mouse );
    setVisible( true );
    canvas.createBufferStrategy( 2 );
    bs = canvas.getBufferStrategy();
    canvas.requestFocus();
    gameThread = new Thread( this );
    gameThread.start();
}

protected void onComponentResized( ComponentEvent e ) {
    Dimension size = getContentPane().getSize();
    int vw = (int)(size.width * appBorderScale);
    int vh = (int)(size.height * appBorderScale);
    int vx = (size.width - vw) / 2;
    int vy = (size.height - vh) / 2;
    int newW = vw;
    int newH = (int)(vw * appWorldHeight / appWorldWidth);
    if( newH > vh ) {
        newW = (int)(vh * appWorldWidth / appWorldHeight);
        newH = vh;
    }
    // center
    vx += (vw - newW) / 2;
    vy += (vh - newH) / 2;
    canvas.setLocation( vx, vy );
    canvas.setSize( newW, newH );
}

protected Matrix3x3f getViewportTransform() {
    return Utility.createViewport(
        appWorldWidth, appWorldHeight, canvas.getWidth(), canvas.getHeight()
    );
}

protected Matrix3x3f getReverseViewportTransform() {
    return Utility.createReverseViewport(
        appWorldWidth, appWorldHeight, canvas.getWidth(), canvas.getHeight()
    );
}
```



```
}

protected Vector2f getWorldMousePosition() {
    Matrix3x3f screenToWorld = getReverseViewportTransform();
    Point mousePos = mouse.getPosition();
    Vector2f screenPos = new Vector2f( mousePos.x, mousePos.y );
    return screenToWorld.mul( screenPos );
}

protected Vector2f getRelativeWorldMousePosition() {
    float sx = appWorldWidth / (canvas.getWidth() - 1);
    float sy = appWorldHeight / (canvas.getHeight() - 1);
    Matrix3x3f viewport = Matrix3x3f.scale( sx, -sy );
    Point p = mouse.getPosition();
    return viewport.mul( new Vector2f( p.x, p.y ) );
}

public void run() {
    running = true;
    initialize();
    long curTime = System.nanoTime();
    long lastTime = curTime;
    double nsPerFrame;
    while( running ) {
        curTime = System.nanoTime();
        nsPerFrame = curTime - lastTime;
        gameLoop( (float)(nsPerFrame / 1.0E9) );
        lastTime = curTime;
    }
    terminate();
}

protected void initialize() {
    frameRate = new FrameRate();
    frameRate.initialize();
}

protected void terminate() {

}

private void gameLoop( float delta ) {
    processInput( delta );
    updateObjects( delta );
    renderFrame();
    sleep( appSleep );
}

private void renderFrame() {
    do {
        do {
```



```
Graphics g = null;
try {
    g = bs.getDrawGraphics();
    g.clearRect( 0, 0, getWidth(), getHeight() );
    render( g );
} finally {
    if( g != null ) {
        g.dispose();
    }
}
} while( bs.contentsRestored() );
bs.show();
} while( bs.contentsLost() );
}

private void sleep( long sleep ) {
try {
    Thread.sleep( sleep );
} catch( InterruptedException ex ) { }
}

protected void processInput( float delta ) {
    keyboard.poll();
    mouse.poll();
}

protected void updateObjects( float delta ) {
}

protected void render( Graphics g ) {
    g.setFont( appFont );
    g.setColor( appFPSColor );
    frameRate.calculate();
    g.drawString( frameRate.getFrameRate(), 20, 20 );
}

protected void onWindowClosing() {
try {
    running = false;
    gameThread.join();
} catch( InterruptedException e ) {
    e.printStackTrace();
}
System.exit( 0 );
}

protected static void launchApp( final SimpleFramework app ) {
    app.addWindowListener( new WindowAdapter() {
        public void windowClosing( WindowEvent e ) {

```



```
        app.onWindowClosing();
    }
});
SwingUtilities.invokeLater( new Runnable() {
    public void run() {
        app.createAndShowGUI();
    }
});
}
}
```

## 5.3 使用简单框架模板

位于 `javagames.util` 包中的 `SimpleFrameworkTemplate`，可以用做后面所有示例的一个起点。该框架隐藏了通用的代码，使我们很容易开发示例中的新代码，如图 5.4 所示。

```
// SimpleFrameworkTemplate.java
public static void main( String[] args ) {
    launchApp( new SimpleFramework
Template() );
}
```

该模板扩展了 `SimpleFramework` 类。在 `main` 方法中，`launchApp()`方法用来启动应用程序。

在这个构造方法中，设置了所有的应用程序属性。在 `initialize()`方法中设置属性的话，有点太晚了，因为在 `initialize()`方法调用之前，有些属性已经用到了。默认值可以接受的任何属性，都可能从构造方法中移除。



不要在 `initialize()`方法中设置属性，在构造函数中设置它们。



图 5.4 简单框架模板

覆盖如下的 5 个方法，以便在游戏循环中执行定制代码。

- `initialize()`



- processInput(float delta)
- updateObjects(float delta)
- render(Graphics g)
- terminate()

如下的示例模板，可以用做后面各章示例的一个基础。

```
package javagames.util;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import javagames.util.SimpleFramework;

public class SimpleFrameworkTemplate extends SimpleFramework {
    public SimpleFrameworkTemplate() {
        appBackground = Color.WHITE;
        appBorder = Color.LIGHT_GRAY;
        appFont = new Font( "Courier New", Font.PLAIN, 14 );
        appBorderScale = 0.9f;
        appFPSColor = Color.BLACK;
        appWidth = 640;
        appHeight = 640;
        appMaintainRatio = true;
        appSleep = 10L;
        appTitle = "FrameworkTemplate";
        appWorldWidth = 2.0f;
        appWorldHeight = 2.0f;
    }
    @Override
    protected void initialize() {
        super.initialize();
    }
    @Override
    protected void processInput( float delta ) {
        super.processInput( delta );
    }
    @Override
    protected void updateObjects( float delta ) {
        super.updateObjects( delta );
    }
    @Override
    protected void render( Graphics g ) {
```



```
        super.render( g );
    }
    @Override
    protected void terminate() {
        super.terminate();
    }
    public static void main( String[] args ) {
        launchApp( new SimpleFrameworkTemplate() );
    }
}
```

## 5.4 资源和延伸阅读

Code and a Coke 2D framework tutorial; see <http://www.cokeandcode.com/main/tutorials/space-invaders-101/>.

Java framework for desktop, iOS, Android, and HTML games; see <http://libgdx.badlogicgames.com/>.

The lightweight Java game library; see <http://lwjgl.org/>.



# 第 6 章

## Vector2f 更新

在继续学习相交测试算法和概念之前，需要注意 `vector` 类有一些更新。本章介绍这些更新。

### 6.1 `inv()`

```
public Vector2f inv() {
    return new Vector2f( -x, -y );
}
```

`inv()`方法返回与当前向量相反的一个向量。一个向量的取反，如图 6.1 所示，是指朝相反方向的一个新向量。

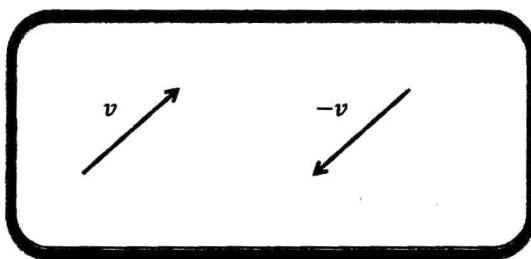


图 6.1 向量取反

### 6.2 `add()`

```
public Vector2f add( Vector2f v ) {
    return new Vector2f( x + v.x, y + v.y );
}
```

add()方法返回一个新的向量，它是当前向量和给定向量的加和，如图 6.2 所示。

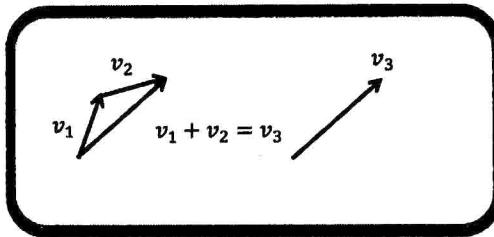


图 6.2 向量相加

### 6.3 sub( )

```
public Vector2f sub( Vector2f v ) {  
    return new Vector2f( x - v.x, y - v.y );  
}
```

sub()方法返回一个新的向量，它是当前向量和给定向量相减的结果，如图 6.3 所示，注意，顺序很重要。新向量的箭头将会指向相减操作中的第一个向量。

有时候，将减法看做是加上一个相反方向的向量，这会更加形象化。如图 6.4 所示， $V_1 - V_2$  和  $V_1 + (-V_2)$  得到相同的结果。

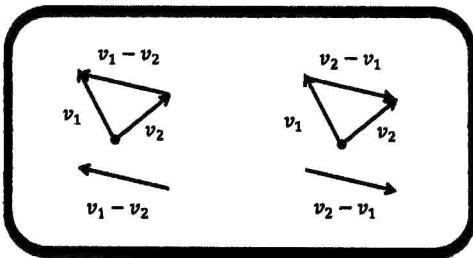


图 6.3 向量相减

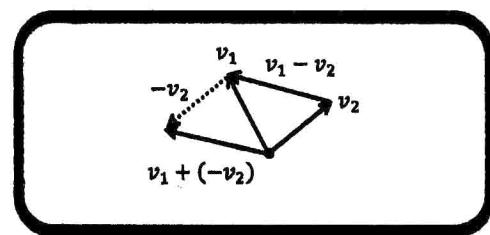


图 6.4 向量减法就是加上相反方向的向量

### 6.4 mul( )

```
public Vector2f mul( float scalar ) {
```

```
    return new Vector2f( scalar * x, scalar * y );
}
```

mul()方法返回一个新的向量，它是向量部分和标量值的乘积。注意，图 6.5 中，负号将向量取反。

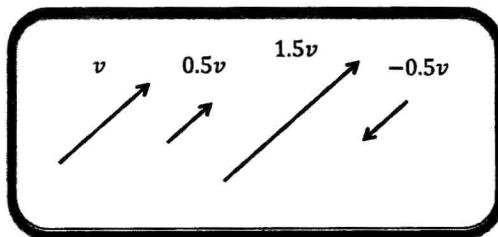


图 6.5 向量乘法

## 6.5 div( )

```
public Vector2f div( float scalar ) {
    return new Vector2f( x / scalar, y / scalar );
}
```

div()方法和乘法相同，也是根据一个标量来进行操作。注意，没有对除以 0 的情况进行检查。

## 6.6 len( )和 lenSqr( )

```
public float len() {
    return (float)Math.sqrt( x * x + y * y );
}
```

要得到一个向量的长度，使用毕达哥拉斯定理，如图 6.6 所示。

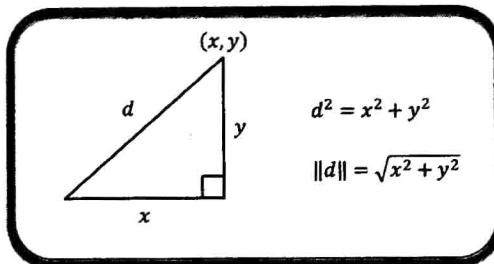


图 6.6 向量的长度

单竖线用来表示绝对值 $|a|$ 。双竖线用来表示一个向量的长度 $\|a\|$ 。

$$\text{If } x > y \text{ then } x^2 > y^2$$

```
public float lenSqr() {  
    return x * x + y * y;  
}
```

由于求平方操作需要占用较长的计算时间，有时候，只是比较长度值的平方更容易一些。

## 6.7 norm( )

```
public Vector2f norm() {  
    return div( len() );  
}
```

norm()返回一个正规化向量，其长度为 1，如图 6.7 所示。很多图形公式都可以使用长度为 1 的向量来进行简化。变量上的“小帽子”（脱字符号）表示这是一个单位向量。

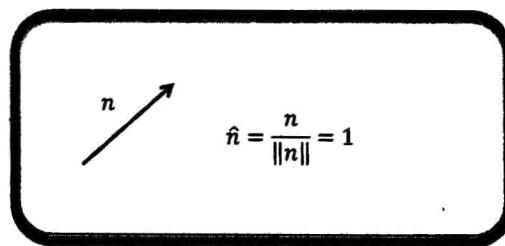


图 6.7 正规化向量

## 6.8 perp( )

```
public Vector2f perp() {  
    return new Vector2f( -y, x );  
}
```

`perp()`方法返回一个新的向量，它是当前向量的垂直向量，如图 6.8 所示。这对于创建一个正规化向量很有帮助，后面各章的很多公式中都会用到这样的正规化向量。注意， $(-y, x)$  和  $(y, -x)$  都会产生一个垂直向量。一个倒过来的 T 表示一个垂直向量。

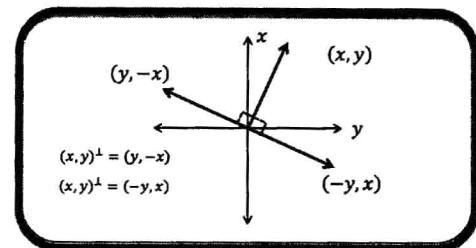


图 6.8 正交向量

## 6.9 dot( )

```
public float dot( Vector2f v ) {  
    return x * v.x + y * v.y;  
}
```

`dot()`方法返回两个向量的点积。这个将两个向量相乘的方法非常有用。如图 6.9 所示，点积将一个向量映射到另一个向量之上，形成其右边的一个三角，在测试对象相交性的时候，这很有用。

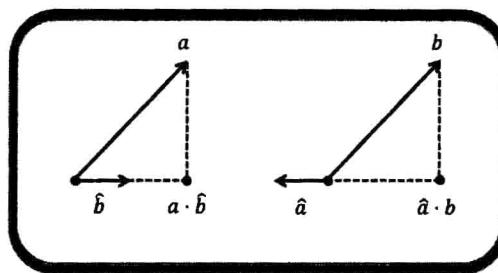


图 6.9 点积

点积的证明如图 6.10 所示，可以表示如下：

$$A * B = \| A \| \| B \| \cos(\theta A - \theta B)$$

$$A * B = AB[\cos(\theta A) \cos(\theta B) + \sin(\theta A) \sin(\theta B)]$$

$$A * B = A \cos(\theta A)B \cos(\theta B) + A \sin(\theta A)B \sin(\theta B)$$

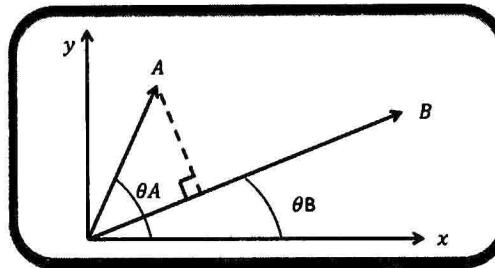


图 6.10 点积的角

来看看笛卡尔坐标转换：

$$A_x = A \cos(\theta A)$$

$$B_x = B \cos(\theta B)$$

$$A_y = A \sin(\theta A)$$

$$B_y = B \sin(\theta B)$$

$$A * B = A_x B_x + A_y B_y$$

点积具有如下的数学特性：

$$A * B = B * A$$

$$C * (A + B) = C * A + C * B$$

$$C(A * B) = CA * B$$

$$A * A^\perp = 0$$

最后一个特性很重要，并且用于很多数学证明的基础。由于两个垂直的向量之间的角度总是为零，因此它们的点积总是为零。

$$A * A^\perp = \| A \| \| A^\perp \| \cos(90^\circ)$$

$$A * A^\perp = \| A \| \| A^\perp \| (0)$$

$$A * A^\perp = 0$$

最后，还有重要的一点是，点积返回一个标量值，而不是一个向量，如图 6.11 所示。要将一个向量投射到另一个向量之上，接受投射的向量必须是一个正规化向量，并且该正

规范化向量需要由点积来缩放。

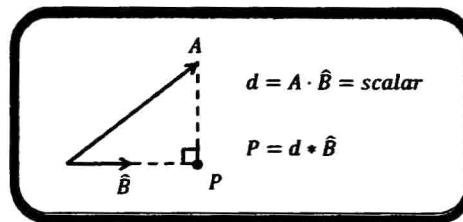


图 6.11 点积标量

## 6.10 angle( )

```
public float angle() {  
    return (float)Math.atan2( y, x );  
}
```

angle()方法返回向量所创建的角度（以弧度为单位），如图 6.12 所示。注意，atan2()方法用来处理  $x$  为 0 的情况，因此，不需要进行额外的检查。

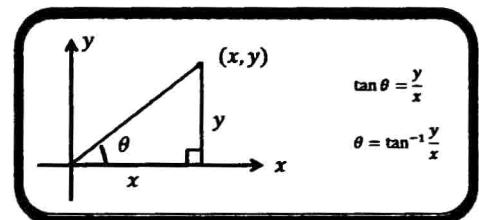


图 6.12 角度

## 6.11 polar( )

```
public static Vector2f polar( float angle, float radius ) {  
    return new Vector2f(  
        radius * (float)Math.cos( angle ),  
        radius * (float)Math.sin( angle )  
    );  
}
```

polar()方法接受以弧度为单位的一个角度值和一个半径，返回用转换后的  $x$  和  $y$  坐标创建的一个向量，如图 6.13 所示。

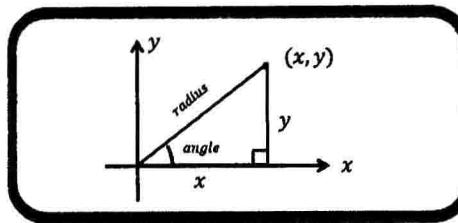


图 6.13 极坐标

## 6.12 `toString()`

```
@Override  
public String toString() {  
    return String.format( "(%s,%s)", x, y );  
}
```

最后的一项更新，帮助把向量值显示为字符串以进行调试。

```
package javagames.util;  
  
public class Vector2f {  
    public float x;  
    public float y;  
    public float w;  
    public Vector2f() {  
        this.x = 0.0f;  
        this.y = 0.0f;  
        this.w = 1.0f;  
    }  
    public Vector2f( Vector2f v ) {  
        this.x = v.x;  
        this.y = v.y;  
        this.w = v.w;  
    }  
    public Vector2f( float x, float y ) {  
        this.x = x;  
        this.y = y;  
        this.w = 1.0f;
```



```
}

public Vector2f( float x, float y, float w ) {
    this.x = x;
    this.y = y;
    this.w = w;
}

public void translate( float tx, float ty ) {
    x += tx;
    y += ty;
}

public void scale( float sx, float sy ) {
    x *= sx;
    y *= sy;
}

public void rotate( float rad ) {
    float tmp = (float)( x * Math.cos( rad ) - y * Math.sin( rad ) );
    y = (float)( x * Math.sin( rad ) + y * Math.cos( rad ) );
    x = tmp;
}

public void shear( float sx, float sy ) {
    float tmp = x + sx * y;
    y = y + sy * x;
    x = tmp;
}

public Vector2f add( Vector2f v ) {
    return new Vector2f( x + v.x, y + v.y );
}

public Vector2f sub( Vector2f v ) {
    return new Vector2f( x - v.x, y - v.y );
}

public Vector2f mul( float scalar ) {
    return new Vector2f( scalar * x, scalar * y );
}

public Vector2f div( float scalar ) {
    return new Vector2f( x / scalar, y / scalar );
}

public Vector2f inv() {
    return new Vector2f( -x, -y );
}

public Vector2f norm() {
    return div( len() );
}

public float dot( Vector2f v ) {
```



```
        return x * v.x + y * v.y;
    }
    public float len() {
        return (float)Math.sqrt( x * x + y * y );
    }
    public float lenSqr() {
        return x * x + y * y;
    }
    public Vector2f perp() {
        return new Vector2f( -y, x );
    }
    public float angle() {
        return (float)Math.atan2( y, x );
    }
    public static Vector2f polar( float angle, float radius ) {
        return new Vector2f(
            radius * (float)Math.cos( angle ),
            radius * (float)Math.sin( angle )
        );
    }
    @Override
    public String toString() {
        return String.format( "(%s,%s)", x, y );
    }
}
```

## 6.13 资源和延伸阅读

Akenine-Moller, Tomas, and Eric Haines, *Real-Time Rendering*, 2<sup>nd</sup> Edition, A K Peters, 2002.

Ericson, Christer, *Real-Time Collision Detection*, Focal Press, 2004.



# 第 7 章

## 相交测试

如果一款游戏中的物体绕着屏幕来回飞，但是却从不相交，这种游戏真是很少见。不管是什么类型的游戏，即便它只是检查鼠标是否悬停在某个物体上，在某些时候，它也是需要相交测试和碰撞检测的。问题是，这些主题可能很复杂。在很多书架上，可以找到整本关于算法和数学的图书，但人们仍然在试图寻找改进算法的方法。大多数人认为巨大的 3D 沙箱游戏中充满了复杂的思路，而（看似）简单的 2D 游戏很容易。在学习完本章之后，你会明白，这种看法显然是一种假象。

### 7.1 多边形中的点的测试

为了确定鼠标指针是否在一个物体之中，需要进行两个操作。首先，需要将屏幕坐标表示的鼠标位置转换为世界坐标。第 5 章中介绍了相关的代码。其次，需要测试物体看看鼠标位置的世界坐标是否位于物体之中。测试一个点是否在一个多边形中，这对于计算机游戏中的各种不同环境来说都很有用。

判断一个点是否位于多边形之中的算法很容易理解，但是，乍一看，有很多概念并不是一目了然的。测试本身很容易说明，但是很难实现。你获取要测试的点，并且沿着  $x$  轴画一条射线。每当这条射线穿过多边形的时候，就计数一次。如果得到的是偶数次，则这个点在多边形之外。如果得到的是奇数次，那么这个点在多边形之中，如图 7.1 所示。

如图 7.1 所示，点 0、2 和 4 在多边形之外，因为它们穿过多边形偶数次。点 1 和 3 在多边形之内，因为它们穿过了奇数次。由于算法需要知道射线是否穿过了多边形的一条边，因此，需要将测试分解为 4 种不同的情况，如图 7.2 所示。

前 3 种情况中，多边形的边线在点的上方、下方或后面，这都会导致没有相交。只有第 4 种情况视为射线穿过了多边形边线。

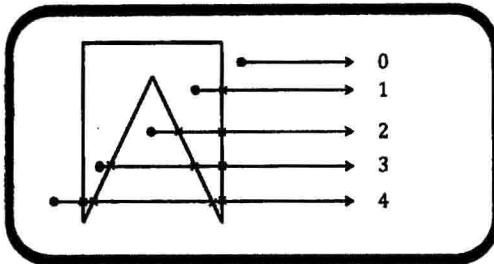


图 7.1 多边形中的点

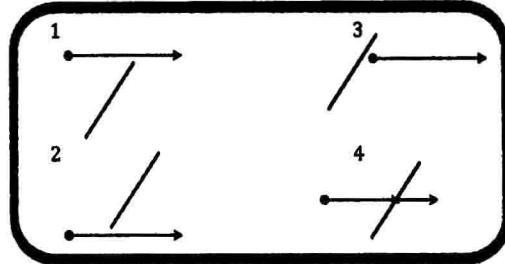


图 7.2 多边形中的点

```
// PointInPolygonExample.java
private boolean pointInPolygon( Vector2f point, List<Vector2f> poly ) {
    boolean inside = false;
1:    if( poly.size() > 2 ) {
2:        Vector2f start = poly.get( poly.size() - 1 );
3:        boolean startAbove = start.y >= point.y;
4:        for( int i = 0; i < poly.size(); ++i ) {
5:            Vector2f end = poly.get( i );
6:            boolean endAbove = end.y >= point.y;
7:            if( startAbove != endAbove ) {
8:                // TBD - test ahead or behind
9:            }
10:           startAbove = endAbove;
11:           start = end;
12:       }
13:   }
14:   return inside;
}
```

pointInPolygon() 算法的步骤如下。

1. 如果多边形没有 3 条以上的边，就不用麻烦测试了。
2. 从最后一个点和第一个点开始。
3. 如果多边形的第一个点在要检查的点的 y 值之上，将 `startAbove` 设置为 `true`。
4. 遍历多边形中的所有的点。
5. 代码获取了多边形中的最后一个点后，从第一个点开始。
6. 如果多边形的第二个点在要检查的点的 y 值之上，将 `endAbove` 设置为 `true`。
7. 如果 `startAbove` 和 `endAbove` 是相等的，那么多边形的两个点要么都在该点之上或都在该点之下，并且没有相交。
8. 保存 `startAbove` 状态。

9. 以终点作为起点。

10. 返回结果。

为了判断相交是在射线的前面还是后面，点斜式方程可以用来求解未知的  $x$  坐标值。

$$\begin{aligned}y - y_1 &= m(x - x_1) \\x - x_1 &= \frac{y - y_1}{m} \\x = x_1 &= \frac{y - y_1}{m} \\where m &= \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}\end{aligned}$$

一旦找到了  $x$  交点值，将其与该点的  $x$  值进行比较。如果交点值小于该点的  $x$  值，交点在该点之后，并且没有发生相交。

```
if( startAbove != endAbove ) {  
    float m = (end.y - start.y) / (end.x - start.x);  
    float x = start.x + (point.y - start.y) / m;  
    if( x >= point.x ) {  
        inside = !inside;  
    }  
}
```

### 7.1.1 多边形特例中的点

只要多边形的边没有相交，这个算法工作得很好。如下的两个多边形有相交的边，并且黑色阴影部分被看做是在多边形之外，而实际上这部分在多边形之内，如图 7.3 所示。

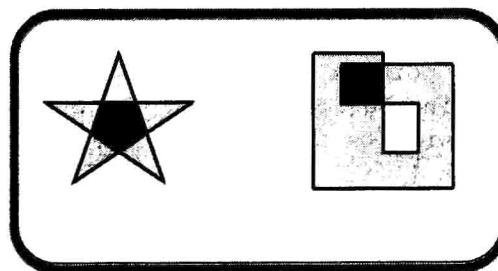


图 7.3 多边形特例中的点

要正确地识别这些重叠的部分，需要统计边线和射线相交的方向。从顶部到底部，每

相交一次，计数增加 1；从底部到顶部，每相交一次，计数减少 1。如果相交的次数之中，顺时针和逆时针相交的次数相同（计数为 0），那么，该点在多边形之外；如图 7.4 所示。

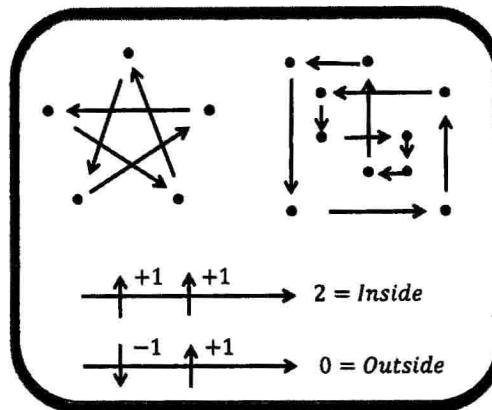


图 7.4 多边形环绕

带环绕和不带环绕的完整算法，如下所示：

```
// PointInPolygonExample.java
private boolean pointInPolygon(
    Vector2f point, List<Vector2f> poly, boolean winding ) {
    int inside = 0;
    if( poly.size() > 2 ) {
        Vector2f start = poly.get( poly.size() - 1 );
        boolean startAbove = start.y >= point.y;
        for( int i = 0; i < poly.size(); ++i ) {
            Vector2f end = poly.get( i );
            boolean endAbove = end.y >= point.y;
            if( startAbove != endAbove ) {
                float m = (end.y - start.y) / (end.x - start.x);
                float x = start.x + (point.y - start.y) / m;
                if( x >= point.x ) {
                    if( winding ) {
                        inside += startAbove ? 1 : -1;
                    } else {
                        inside = inside == 1 ? 0 : 1;
                    }
                }
            }
            startAbove = endAbove;
            start = end;
        }
    }
}
```



```
        }
    }
    return inside != 0;
}
```

### 7.1.2 多边形示例中的点

位于 `javagames.intersection` 包中的 `PointInPolygonExample`, 如图 7.5 所示, 它针对所有模板代码使用了 `SimpleFramework` 类。`poly` 列表保存了通过在屏幕上点击而创建的多边形。`polyCpy` 列表保存了转换为屏幕坐标的所有多边形。多边形内的所有点, 都放在了 `inside` 列表中; 多边形之外的所有的点都放在 `outside` 列表中。鼠标位置转换为世界坐标。如果鼠标在多边形中, `selected` 变量设置为 `true`。如果该算法在测试点的时候应该使用环绕, `winding` 变量为 `true`。

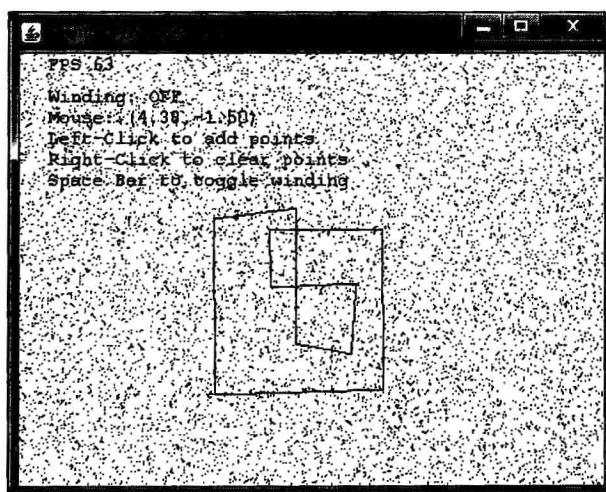


图 7.5

`Utility` 类中加入了几个新的方法来绘制多边形。这些代码和前面示例中所使用的代码相同, 但是, 将其添加到 `Utility` 类中, 比在所有的示例中都进行复制和粘贴要更加容易。

```
package javagames.util;
import java.awt.Graphics;
import java.util.List;
public class Utility {
// other methods ...
public static void drawPolygon(
```



```
Graphics g, Vector2f[] polygon ) {  
    Vector2f P;  
    Vector2f S = polygon[ polygon.length - 1 ];  
    for( int i = 0; i < polygon.length; ++i ) {  
        P = polygon[i];  
        g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );  
        S = P;  
    }  
}  
public static void drawPolygon(  
    Graphics g, List<Vector2f> polygon ) {  
    Vector2f S = polygon.get( polygon.size() - 1 );  
    for( Vector2f P : polygon ) {  
        g.drawLine( (int)S.x, (int)S.y, (int)P.x, (int)P.y );  
        S = P;  
    }  
}
```

在 `processInput()`方法中，鼠标位置转换为世界坐标（在第 5 章中介绍过）。空格键切换多边形环绕。点击鼠标左键，添加多边形点；点击鼠标右键，清除多边形。

如果鼠标位于多边形之中，用 `updateObjects()`方法设置选中的点。然后，根据多边形测试随机的点，并将其放入到相应的列表中。前面介绍的 `pointInPolygon()`方法，用来判断点相对于多边形的位置。`render()`方法显示了环绕、世界坐标中的鼠标位置，以及控制说明。如果鼠标位于多边形之中，多边形绘制成绿色；否则，将其绘制成蓝色。多边形中所有随机生成的点，都绘制为绿色；而多边形之外的所有点，都绘制为红色。

```
package javagames.intersection;  
import java.awt.*;  
import java.awt.event.*;  
import java.util.*;  
import java.util.List;  
import javagames.util.*;  
public class PointInPolygonExample extends SimpleFramework {  
    private static final int MAX_POINTS = 10000;  
    private ArrayList<Vector2f> poly;  
    private ArrayList<Vector2f> polyCpy;  
    private ArrayList<Vector2f> inside;  
    private ArrayList<Vector2f> outside;  
    private Vector2f mousePos;  
    private boolean selected;  
    private boolean winding;
```



```
public PointInPolygonExample() {
    appWidth = 640;
    appHeight = 640;
    appTitle = "Point In Polygon Example";
    appBackground = Color.WHITE;
    appFPSColor = Color.BLACK;
}
@Override
protected void initialize() {
    super.initialize();
    // polygon points and lists of point inside
    // and outside the polygon
    poly = new ArrayList<Vector2f>();
    polyCpy = new ArrayList<Vector2f>();
    inside = new ArrayList<Vector2f>();
    outside = new ArrayList<Vector2f>();
    mousePos = new Vector2f();
}
@Override
protected void processInput( float delta ) {
    super.processInput( delta );
    mousePos = getWorldMousePosition();
    // draw polygon for algorithm testing
    if( keyboard.keyDownOnce( KeyEvent.VK_SPACE ) ) {
        winding = !winding;
    }
    if( mouse.buttonDownOnce( MouseEvent.BUTTON1 ) ) {
        poly.add( mousePos );
    }
    if( mouse.buttonDownOnce( MouseEvent.BUTTON3 ) ) {
        poly.clear();
    }
}
@Override
protected void updateObjects( float delta ) {
    super.updateObjects( delta );
    // see if the mouse is inside the polygon
    selected = pointInPolygon( mousePos, poly, winding );
    // test random points against the polygon
    Random rand = new Random();
    inside.clear();
    outside.clear();
    for( int i = 0; i < MAX_POINTS; ++i ) {
```



```
    float x = rand.nextFloat() * 2.0f - 1.0f;
    float y = rand.nextFloat() * 2.0f - 1.0f;
    Vector2f point = new Vector2f( x, y );
    if( pointInPolygon( point, poly, winding ) ) {
        inside.add( point );
    } else {
        outside.add( point );
    }
}
}
private boolean pointInPolygon( Vector2f point, List<Vector2f> poly,
    boolean winding ) {
    // point in polygon algorithm
    int inside = 0;
    if( poly.size() > 2 ) {
        Vector2f start = poly.get( poly.size() - 1 );
        boolean startAbove = start.y >= point.y;
        for( int i = 0; i < poly.size(); ++i ) {
            Vector2f end = poly.get( i );
            boolean endAbove = end.y >= point.y;
            if( startAbove != endAbove ) {
                float m = (end.y - start.y) / (end.x - start.x);
                float x = start.x + (point.y - start.y) / m;
                if( x >= point.x ) {
                    if( winding ) {
                        inside += startAbove ? 1 : -1;
                    } else {
                        inside = inside == 1 ? 0 : 1;
                    }
                }
            }
            startAbove = endAbove;
            start = end;
        }
    }
    return inside != 0;
}
@Override
protected void render( Graphics g ) {
    super.render( g );
    // render instructions
    g.drawString( "Winding: " + (winding?"ON":"OFF"), 20, 35 );
    String mouse =
```