第20章　控制矢量图形

由于矢量图形演员是Director 7的新添内容，因此它们需要一整套新的 Lingo属性和命令。实际上，这些属性中许多是从在 Director 6.5中引入的Flash演员那里借用过来的,其中的一个重要原因是由于矢量图形演员使用了 Flash Assert Xtra 中的Flash引擎。本章将探讨Flash演员和矢量图形演员的Lingo。要想使用这些演员，就要理解这些内容。

20.1　使用Flash演员的Lingo

毋庸质疑，Flash演员是最复杂的演员。毕竟，它们是在 Macromedia Flash中创建的，它本身是一个独立的程序。 Flash创建了大多数用于因特网的基于矢量图形的动画。这个程序本身就值得写很多本书，而且现在也有几本可供我们选择了。

同Director影片一样，Flash演员也是由很多帧构成的。每帧包括不同的元素，如图形、位图、按钮和声音等。因此，它不是像位图那样的静止的素材，而是与数字视频文件类似的基于时间的素材。大多数用于Flash影片的Lingo语言反映了这个事实。

下面这些属性可以在Flash影片在舞台上播放时用来控制播放速度：

■ playBackMode(播放模式) —— 我们可以有三种选项： #normal、#lockstep和#fixed。#normal选项告诉Flash影片应当使用在 Flash中创建它时所使用的设置。 #Lockstep设置使得Flash中的一帧对应Director影片的一帧而播放。 #Fixed设置采用演员的 fixedRate属性来确定Flash 影片的帧速度。

■ fixedRate(固定速度) —— 只有在playBackMode为#fixed时才设置。这样就可以定义 Flash影片的帧速度。甚至在播放时还可以改变。

■ frameRate(帧速度) —— 这个属性能够告诉我们Flash 影片的原始帧速度。

■ frameCount(帧计数器) —— 这个属性能够告诉我们Flash影片中的帧的数量。

也可以用Lingo在Flash影片中进行浏览。请看下面的命令：

■ goToFrame —— 这个命令允许我们指定 Flash影片中的帧编号和帧标签。例如：goToFrame(sprite 5, "intro")。角色必须包含一个Flash演员。

■ findLable —— 这个函数取一个角色编号和帧标签为参数，并试图返回 Flash影片中的带有这个标签的帧的编号。如果没有发现，就会返回 0。

■ pauseAtStart —— 可以设置为TRUE或FALSE。该属性确定Flash影片出现在舞台上时是否自动开始播放。

■ playing —— 这个属性可以告诉我们Flash影片现在是否正在播放。

■ Rewind —— 这个命令取角色的编号作为它的一个参数。它可以使 Flash影片返回到第1帧。

■ Stop —— 这个命令可以使指定的角色中的 Flash影片停止播放。

■ Play —— 这个命令可以使指定的角色中的 Flash影片继续播放。

我们也可以控制Flash 演员与Flash影片的特性有多么相像。由于 Flash 影片有自己的按钮

和动作，有些时候，当把它们用在 Director中时，我们可能想让 Flash 演员中的这些元素不起作用。

■ actionEnable —— 确定在Flash 角色中或演员中的任何动作是否有效。

■ buttonEnable —— 确定在Flash角色中的按钮是否有效。

我们还可以决定何时让鼠标事件(例如点击和拖动)被传送给与角色相关联的 Lingo行为。

■ clickMode —— 角色或演员的这个属性可以被设置为下列三者之一： #boundingBox、#opaque和#object。#boundingBox选项表示在整个角色矩形区域内都能够检测点击及其他鼠标事件。#opaque选项表示只有当光标落在 Flash演员的不透明部位时才能检测到点击事件。#object选项表示只有当光标落在 Flash演员的带填充色的图形上时才能检测到点击事件。只有当角色被设置为Background Transparent油墨时，#opaque选项才有效。

■ eventPassMode —— 确定是否把点击操作传递给 Lingo行为。有四种模式：#passAlways、#passButton、#passNotButton和#passNever。#passAlways为缺省设置。

另外两个函数可以告诉我们某点是否落在 Flash 影片的角色上：

■ hitTest —— 这个函数有两个参数：角色和点。该点应与角色的位置有关。根据该点的位置，它可以返回#background、#normal和#button。#normal意味着它在Flash影片中的某个图形上。

■ mouseOverButton —— 如果光标位于指定的Flash影片角色的一个按钮上，则返回 TRUE。

参见第 7 章“ 矢量图形演员 ”里的 7.3节“ Flash演员 ”，可以获得在Director里使用Flash演员的背景资料。

20.2 使用矢量图形的Lingo

矢量图形演员与 Flash演员有一个重要的区别：它们是静止的图像，而不是基于时间的素材。然而，与 Flash演员不同，它们可以完全由 Lingo控制。我们甚至可以从零开始用 Lingo创建一个矢量图形，并设计成我们所需要的样子。

提示 由于矢量图形演员是用Director中的Flash引擎绘制的，因而Flash 演员的许多属性可供矢量图形使用。反之亦然。

组合的矢量图形属性能够使我们改变矢量图形的每一个方面。在这里全部列出：

■ antiAlias —— 确定演员是否为无锯齿。当闭该属性时，演员的绘制会快一些，但线条看起来不是很光滑。

■ backgroundColor —— 演员(而不是角色的)背景色。

■ broadcastProps —— 确定对演员所做的改动是否立即显示在舞台上。如果不立即显示，角色只有在演员离开舞台后再次出现在舞台上时，才会显示所发生的变化。

■ centerRegPoint —— 如果设置为TRUE，当角色的尺寸被重新设置后，演员的套准点会自动改变。当演员显示在舞台上时，如果要采用 Lingo改变矢量图形，应当先将此属性设置为FALSE，以防止角色发生跳跃。

■ closed —— 确定矢量图形中的第一个点和最后一个点是否相接。如果要填充矢量图形，必须将其设置为TRUE。

■ defaultRect —— 这是一个可设置属性，可以与 defaultRectMode共同作用，来改变使用该矢量图形演员的新角色的缺省矩形。

■ defaultRectMode —— 这个属性可以被设置为#flash或#fixed。#flash设置把所有使用该演员的新角色设置为该演员的正常矩形。#fixed模式则采用defaultRect属性设置角色的初始矩形。这个设置也会影响到任何一个现存的且未经拉伸的角色。

■ directToStage —— 确定演员是否忽略角色的油墨效果而绘制在所有的其他角色之上。以这种方式绘制演员可以改善运行性能。

■ endColor —— 矢量图形演员的渐变填充色的目标色。使用 rgb或paletteIndex 结构对其进行设置。fillMode必须被设置为gradient，并且closed属性必须为TRUE。

■ fillColor —— 如果fillMode被设置为#solid，该颜色为矢量图形演员内部区域的颜色；如果fillMode被设置为#gradient，该颜色就为渐变的起始色。采用 rgb或paletteIndex结构来对此进行设置。closed属性必须被设置为TRUE。

■ fillCycles —— 对一个fillMode设置为#gradient的矢量图形演员填充的圈数，这个数字应该为1～7。

■ fillDirection —— 用度表示的填充方向。 fillMode必须被设置为 #gradient，而且gradientType应被设置为#linear。

■ fillMode —— 可以被设置为#none、#solid或#gradient。

■ fillOffset —— 这个属性是一个点，对应于水平和垂直两个方向的渐变填充的位移。只有当fillMode被设置为#gradient时才有效。

■ fillScale —— 这与矢量图形编辑窗口中的“ spread(扩展) ”相对应。fillMode必须被设置为#gradient时才有效。

■ flashRect —— 作为演员的矢量图形演员的初始尺寸，而不是作为角色。

■ gradientType —— 可以被设置为#liner或#radial。只有当fillMode被设置为#gradient才能工作。

■ originMode —— 这是顶点和角色中心之间的关系。可以被设置为 #center、#topLeft或#point。#center选项使得顶点与演员的中心相关， #topLeft选项与左上角相关。#point选项采用originPoint属性。在影片播放时，如果我们想调整某一个顶点，应将originMode设置为#center。

■ originPoint —— 此点指的是顶点和演员位置之间的关系。只有将 originMode设置为#point，才可用这个值。我们也可以使用originH和originV属性。

■ scale —— 可以通过此属性设置演员尺寸，例如采用列表 [1.000,1.000],第一项为水平尺寸,第二项为垂直尺寸。这是对角色进行拉伸的另一种办法。

■ scaleMode —— 这等价于矢量图形属性对话框中的演员属性。可以被设置为 #showAll、#noBorder、#exactfit、#noScale和#autoSize。我们也可以将这些属性用作含有矢量图形的角色的属性。

■ strokeColor —— 矢量图形的边线的颜色。strokeWidth必须大于零才有效。

■ strokeWidth —— 矢量图形演员的边线的宽度。

■ vertexList —— 矢量图形的主要属性。它是构成图形状的全部的点的列表。

■ viewPoint —— 这个点使我们能够改变出现在角色正中间的矢量图形的点。

■ viewScale —— 在舞台上缩放矢量图形的另一种方式。

可以看到，在这里有许多矢量图形的属性。这甚至不包括那些同样作用于矢量图形的角色属性，如rotation、flipH、flipV和skew等。

任何矢量图形演员的关键属性是 vertexList。用信息窗口查看一个例子，会帮助我们理解它是如何起作用的。建立一个新的矢量图形并在其中绘制一个矩形。然后，使用信息窗口来查看vertexList。

```
put member(1).vertexList
-- [[#vertex: point(-104.0000, -40.0000)], [#vertex: point(104.0000, -40.0000)],¬
[#vertex: point(104.0000, 41.0000)], [#vertex: point(-104.0000, 41.0000)]]
```

vertexList是由列表构成的列表。每个小列表是只有一个属性 #vertex的属性列表。#vertex的值是一个点。每个点对应着矢量图形的一个顶角。

我们可以用几种方式改变vertexList。Addvertex、deleteVertex和moveVertex命令使我们能够不使用演员属性就可以实现这个目的。例如，要加一个新的顶点，只要采用下面一个命令：

```
addVertex(member(1),3,point(0,0))
```

这个命令在第二个点和第三个点之间加一个点(　0，0)。我们可以通过使用相对点和moveVertex命令移动已经存在的顶点：

```
moveVertex(member(1),3,100,10)
```

这个命令使第三个顶点向右移动 10个像素。我们可以删除一个顶点，如：

```
deleteVertex(member(1),3)
```

如果不采用这些命令，我们可以直接改变全部顶点的列表。下面一系列命令将第三个顶点向右移动10个像素。

```
vl = member(1).vertexList
v = vl[3].vertex
v = v + point(10,0)
vl[3].vertex = v
member(1).vertexList = vl
```

这种方法比仅仅使用 moveVertex要复杂得多。然而，在许多情况下，重新设置整个顶点列表的确有实际意义。如果我们用 Lingo来从零开始创建一个矢量图形，然后我们想用另一个来代替它，但形状稍有些不同，我们可以用同一个处理程序来创建这两个图形。这个处理程序使用参数使得两个图形不同。它不是试图指出哪些点不同，而只是每次替换整个顶点列表。测试表明两种方法在绘制速度上差别不大。

提示　除了#vertex以外，vertexList实际上还有两个元素。它们是#handle1和#handle2。它们也是点。然而，它们对应于点的曲线控制柄。它们与我们在矢量图形编辑窗内编辑某一点时看到的控制柄是相同的。它们在 vertexList中的值是与真正的#vertex点相关的点。

20.3 用Lingo建立矢量图形

通过简单地对一个矢量图形演员的 vertexlist属性进行设置，我们可以用 Lingo创建各种各样我们感兴趣的图形。例如，下面的处理程序创建了一个新的、简单的线条：

```
on makeLine
  mem = new(#vectorShape)
  mem.name = "Line"
  mem.vertexList = [[#vertex: point(0,0)], [#vertex: point(200,100)]]
end
```

这个处理程序使用new命令来创建一个新演员，并为它命名，然后将它的 vertexList设置为简单的两个点。我们可以用strokewidth和strokeColor来对线宽和颜色进行设置。

下面有一个更复杂的处理程序，它用 sin函数创建一条曲线。图 20-1给出了这个处理程序结果，它已被放在了舞台上。

```
on makeSine
  mem = new(#vectorShape)
  mem.name = "Sine"

  list = []
  repeat with x = -100*pi() to 100*pi()
    y = sin(float(x)/100.0)*100
    add list, [#vertex: point(x,y)]
  end repeat
  mem.vertexList = list
end
```

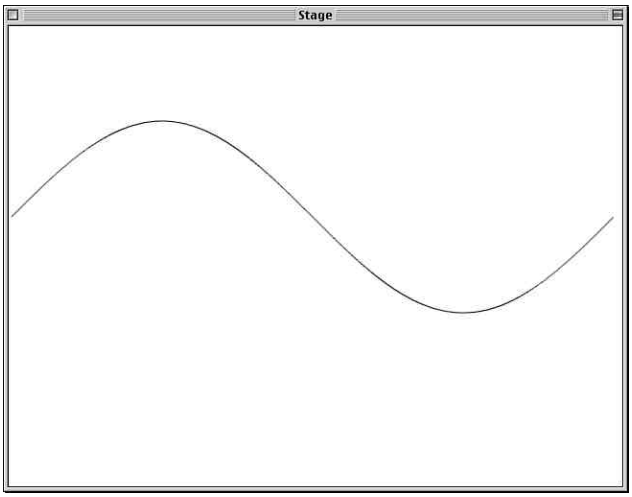

图20-1 这条正弦曲线是由Lingo创建的一个独立的矢量图形演员

要创建一个圆只要多做很少一些工作。与采用矢量图形编辑窗创建的圆不同，这个圆由63个点组成并用线将它们连接起来。尽管它是由短线组成的，但结果看起来很圆。图 20-2给出了结果。

```
on makeCircle
  mem = new(#vectorShape)
  mem.name = "Circle"

  radius = 100

  list = []
  repeat with angle = -10*pi() to 10*pi()
    x = cos(float(angle)/10.0)*radius
    y = sin(float(angle)/10.0)*radius
    add list, [#vertex: point(x,y)]
  end repeat
  mem.vertexList = list
  mem.closed = TRUE
end
```

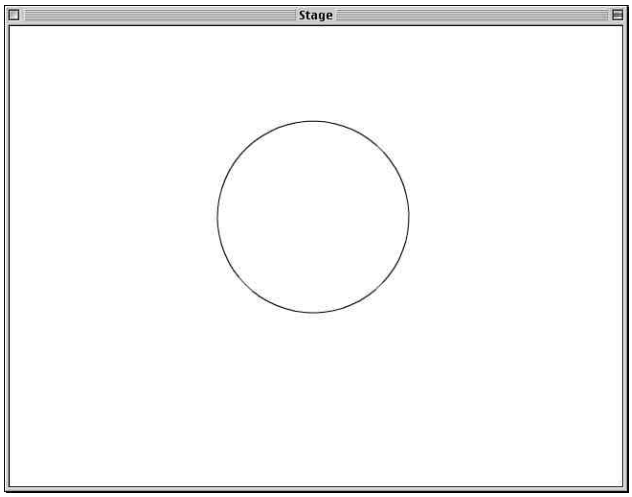

图20-2 用Lingo创建的由一个独立矢量图形中的63条线构成的圆

注意，矢量图形演员的 closed属性要设置为 TRUE。这使得我们也可以将 fillMode属性设置为#solid，并且在fillColor属性中设置我们所要填充的颜色。我们也可以采用渐变色，通过设置fillMode为#gradient，并且设置所有填充属性来指定颜色、类型、方向、尺寸及渐变色填充的类型。

简单的图形无须采用 Lingo矢量图形创建。复杂的图形，例如多边形等才能显示采用Lingo创建矢量图形的优势。下面有一个行为，它把角色中的一个矢量图形设置为多边形。我们可以选择多边形边的边数。

```
property pNumPoints, pRadius

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pNumPoints, [#comment: "Number of Points",¬
    #format: #integer, #default: 5]
  addProp list, #pRadius, [#comment: "Radius",¬
    #format: #integer, #default: 100]
  return list
end

on beginSprite me
  mem = sprite(me.spriteNum).member

  -- how many degrees apart is each point
  angleDiff = 360/pNumPoints

  -- build vertex list
  list = []
  repeat with angle = 0 to pNumPoints-1
    p = circlePoint(angle*angleDiff,pRadius)
    add list, [#vertex: p]
  end repeat

  -- set the member
  mem.vertexList = list
```

```
  mem.closed = TRUE
end

-- the following handler returns the point on any circle
-- given the angle and radius
on circlePoint angle, radius
  a = (float(angle-90)/360.0)*2.0*pi()
  x = cos(a)*radius
  y = sin(a)*radius
  return point(x,y)
end
```

图20-3中显示了如何使用这种行为。它把角色所使用的演员重新设置为多边形。如果在多个角色中使用相同的演员，这些行为会相互干扰。取而代之的是创建一个矢量图形的多个拷贝并将每一个只放置在舞台上一次。最初的矢量图形可以是任意的，例如小的矩形或圆。这个行为重新对角色的 vertexList进行设置，并确认它是闭合的。然而，填充色和类型仍保持前面的设置。

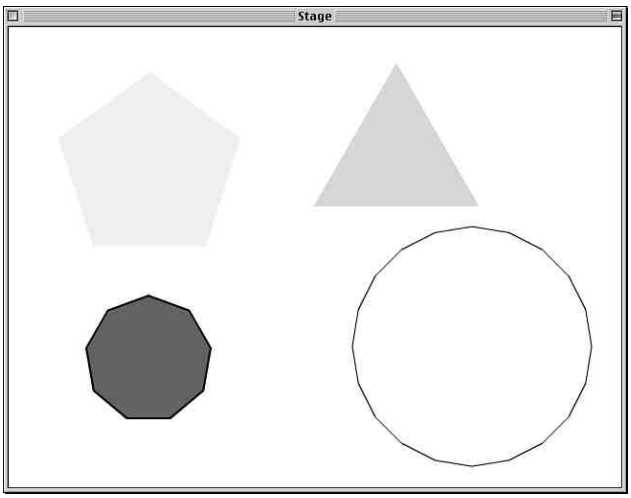

图20-3 这些多边形是用多边形行为建立的，它可以把任意矢量图形塑造成一个多边形

对此行为稍加修改，可以得到一个能够创建星形图形(而不是多边形)的处理程序。它只要交替地设置外部点和内部点就可以了。图 20-4给出了这个行为的结果。

```
property pNumPoints, pRadius

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pNumPoints, [#comment: "Number of Points",¬
    #format: #integer, #default: 5]
  addProp list, #pRadius, [#comment: "Radius",¬
    #format: #integer, #default: 100]
  return list
end

on beginSprite me
  mem = sprite(me.spriteNum).member
```

```
  -- how many degrees apart is each point
  angleDiff = 360/pNumPoints

  -- build vertex list
  list = []
  repeat with starPoint = 0 to pNumPoints-1

    -- outer point location
    p = circlePoint (starPoint*angleDiff,pRadius)
    add list, [#vertex: p]

    -- inner point location
    p = circlePoint((starPoint+.5)*angleDiff,pRadius*.5)
    add list, [#vertex: p]
  end repeat

  -- set the member
  mem.vertexList = list
  mem.closed = TRUE
end

-- the following handler returns the point on any circle
-- given the angle and radius
on circlePoint angle, radius
  a = (float(angle-90)/360.0)*2.0*pi()
  x = cos(a)*radius
  y = sin(a)*radius
  return point(x,y)
end
```

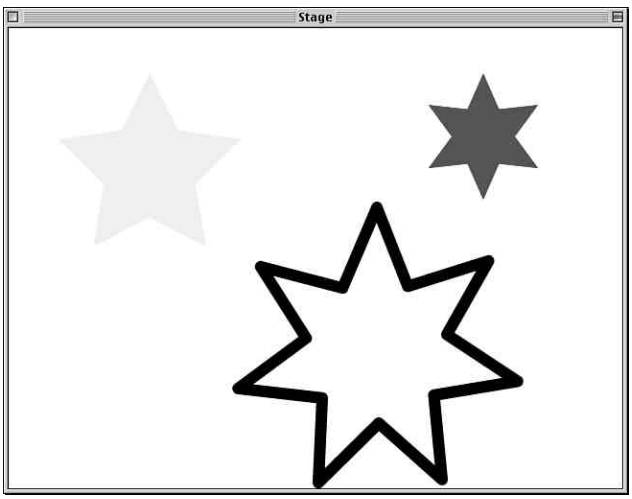

图20-4 星形图形由星形行为建立，它可以把任意矢量图形塑造成星形

前面的处理程序和行为都只把图形绘制一次，通常都使用 on beginSprite，并把矢量图形扔在了那里。然而， Lingo很容易就能在每帧重新绘制一次矢量图形，以创建活动的矢量图形。

下面一个简单的绘制了一个任意形状的例子。该行为采用 20个任意点绘制了一个矢量图

形。结果凌乱潦草。然而，它在每一帧上都重新绘制，得到了一种不同寻常的动画效果。在CD-ROM上查看这个影片，可以看到它的活动效果。

```
on exitFrame me
  mem = sprite(me.spriteNum).member

  list = []
  repeat with i = 1 to 20
    x = random(100)
    y = random(100)
    add list, [#vertex: point(x,y)]
  end repeat
  mem.vertexList = list
end
```

在星形图形的行为里也可以采用这种思路。星星是会闪烁的。下面的行为在每一帧都重新绘制星星，但采用了一些随机的轻微拉伸。如果使用了正确的颜色和尺寸，结果将得到带有动画效果的闪烁的星星。

```
property pNumPoints, pRadius, pPointToMove, pPointMoveDiff, pPointMoveAmount,¬
pTwinkleSpeed, pTwinkleAmount

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pNumPoints, [#comment: "Number of Points",¬
    #format: #integer, #default: 5]
  addProp list, #pRadius, [#comment: "Radius",¬
    #format: #integer, #default: 25]
  addProp list, #pTwinkleSpeed, [#comment: "Twinkle Speed",¬
    #format: #integer, #default: 1]
  addProp list, #pTwinkleAmount, [#comment: "Twinkle Amount",¬
    #format: #integer, #default: 3]
  return list
end

on beginSprite me
  moveNewPoint(me)
  mem = sprite(me.spriteNum).member
  mem.centerRegPoint = FALSE
  mem.originMode = #center
  mem.closed = TRUE
end

-- this handler decides which new point of the star
-- to twinkle
on moveNewPoint me
  repeat while TRUE
    r = random(pNumPoints)
    if r <> pPointToMove then exit repeat
  end repeat
  pPointToMove = r
  pPointMoveDiff = 0
  pPointMoveAmount = pTwinkleSpeed
end

on exitFrame me
```

```
  mem = sprite(me.spriteNum).member

  -- how many degrees apart is each point
  angleDiff = 360/pNumPoints

  -- build vertex list
  list = []
  repeat with starPoint = 1 to pNumPoints

    -- move twinkling point in or out
    if starPoint = pPointToMove then
      pPointMoveDiff = pPointMoveDiff + pPointMoveAmount
      if pPointMoveDiff > pTwinkleAmount then pPointMoveDiff = -pTwinkleSpeed
      if pPointMoveDiff <= 0 then moveNewPoint
      p = circlePoint(starPoint*angleDiff,pRadius+pPointMoveDiff)
    else

      -- keep non-twinkling point normal
      p = circlePoint(starPoint*angleDiff,pRadius)
    end if

    add list, [#vertex: p]
    p = circlePoint((starPoint+.5)*angleDiff,pRadius*.5)
    add list, [#vertex: p]
  end repeat

  -- set the member
  mem.vertexList = list
end

-- the following handler returns the point on any circle
-- given the angle and radius
on circlePoint angle, radius
  a = (float(angle-90)/360.0)*2.0*pi()
  x = cos(a)*radius
  y = sin(a)*radius
  return point(x,y)
end
```

其结果与前面的星形图形行为很相像，但每个点都活动了。它们首先从中心向外移动一点点，然后移回原地。

更有戏剧性的行为要使用顶点的控制柄。由于这些控制柄控制的是曲线进入顶点和离开顶点时的情形，操纵它们难度是很大的。改变这些控制柄很容易，但让它们做我们想做的事情就是另一回事了。即使是多年使用矢量图形编辑软件的插图设计者有时也说不清如何使用控制柄。它们只能靠直觉来使用。这些控制柄的背后是数学计算，但其复杂性超出了本书的范围。

下面的程序是一个行为，它用#handle1为某个矢量图形底部的许多点制造了曲线效果，然后把这些顶点向下移动，得到一种“幕布”或“流血”的效果。图 20-5是动画中的一步。

```
property pNumPoints, pRadius, pVlist

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pNumPoints, [#comment: "Number of Points",¬
```

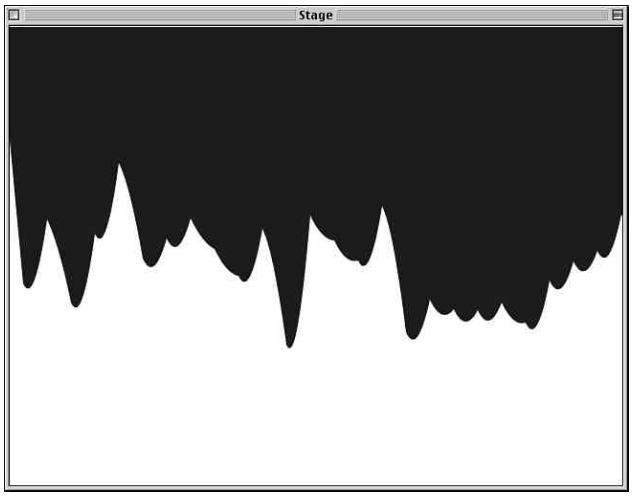

图20-5 “流血”的效果是用矢量图形和一个把顶点向下移的Lingo行为实现的

```
    #format: #integer, #default: 25]
  addProp list, #pRadius, [#comment: "Radius",¬
    #format: #integer, #default: 12]
  return list
end

on beginSprite me
  pVlist = []

  -- space between drips
  spacing = 640/pNumPoints

  -- add top and sides
  add pVlist, [#vertex: point(640+spacing,0)]
  add pVlist, [#vertex: point(640+spacing,0)]
  add pVlist, [#vertex: point(0-spacing,0)]

  -- add drip spots along bottom
  repeat with i = 0 to pNumPoints
    add pVlist, [#vertex: point(i*spacing,0), ¬
#handle1: point(spacing/2,spacing)]
  end repeat

  -- set member
  mem = sprite(me.spriteNum).member
  mem.vertexList = pVlist
  mem.centerRegPoint = FALSE
  mem.originMode = #center
  mem.closed = TRUE
end

on exitFrame me
  -- change 20 vertex points at a time
  repeat with i = 1 to 20
    r = random(pNumPoints+1)+3
```

```
    pVlist[r][#vertex] = pVlist[r][#vertex] + ¬
      point(0,random(pRadius))
  end repeat
  sprite(me.spriteNum) .member.vertexList = pVlist
end
```

另一种效果使用了#handle1和#handle2属性，引导一条由许多点构成的曲线沿着直线移动。在每一帧，曲线的角度会发生变化，使得点像波浪一样地滚动。结果如图 20-6所示，是一个类似于波浪的图。

```
property pNumPoints, pRadius, pList, pOffset, pAngle

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pNumPoints, [#comment: "Number of Points",¬
    #format: #integer, #default: 25]
  addProp list, #pRadius, [#comment: "Radius",¬
    #format: #integer, #default: 12]
  return list
end

on beginSprite me
  pOffset = 0
  pAngle = 0
end

on exitFrame me
  pList = []
  spacing = 680/pNumPoints
  pOffset = pOffset + 2
  if pOffset > spacing then pOffset = 0

  -- create bottom and sides
  add pList, [#vertex: point(680+spacing,0)]
  add pList, [#vertex: point(680+spacing,100)]
  add pList, [#vertex: point(0-spacing,100)]
  - - add wave points
  repeat with i = 0 to pNumPoints

    - - move the waves
    pAngle = pAngle - 1
    if pAngle < -90 then pAngle = 90

    - - get the handle
    h = circlePoint(pAngle,pRadius)
    h2 = circlePoint (pAngle+180,pRadius)

    add pList, [#vertex: point(i*spacing-pOffset,0), ¬
#handle1: h, #handle2: h2]
  end repeat

  -- set the member
  mem = sprite(me.spriteNum).member
  mem.vertexList = pList
  mem.centerRegPoint = FALSE
```

```
  mem.originMode = #center
  mem.closed = TRUE
end

--the following handler returns the point on any circle
--given the angle and radius
on circlePoint angle, radius
  a=(float(angle-90)/360.0)*2.0*pi()
  x=cos(a)*radius
  y=sin(a)*radius
  returnpoint(x,y)
end
```

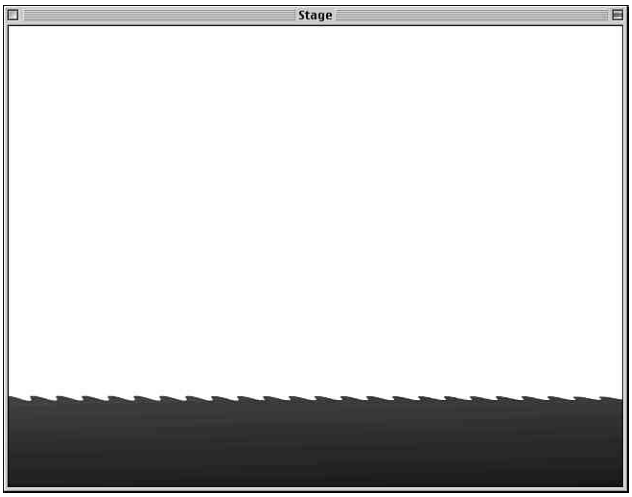

图20-6 改变处理程序中每个顶点的位置，创建动画的波浪效果

20.4 学习高级的矢量图形技巧

矢量图形演员与位图的区别之一在于，对于矢量图形，我们可以确切地知道某一个点是位于矢量图形的里面还是外面；使用位图及其 the rollover属性虽然也能做到这一点，但仅有当油墨设置为Matte，且鼠标落在我们想要检测的区域时才能实现。

bitTest函数使我们能够指定一个角色和一个舞台位置。它返回的数值可以是 #backgroud、#normal和#button中的任意一个。最后一个值只能由 Flash影片返回，而不能由矢量图形返回。然而，另外两个值可以用来确定任何一个点是在矢量图形的外部还是内部，无论矢量图形的形状有多么复杂。

下面的程序是利用了此特征的行为。它假设在角色 1中有一个矢量图形，该行为不能移动这个角色。它会搜索按键操作，并适当地移动角色。如果它发现新的位置是在那个角色的里面，就不允许移动发生。

```
property px, py

on beginSprite me
  -- get initial location
  px = sprite(me.spriteNum).locH
  py = sprite(me.spriteNum).locV
end
```

```
on exitFrame me

  -- assume x doesn′t change
  newx = px

  -- see if it does
  if keyPressed(123) then newx = px - 1
  if keyPressed(124) then newx = px + 1

  -- see if new x will hit the shape
  if hitTest(sprite(1),point(newx,py)) < > #normal then
    px = newx
  end if

  -- assume y doesn't change
  newy = py

  -- see if it does
  if keyPressed(125) then newy = py + 1
  if keyPressed(126) then newy = py - 1

  -- see if new y will hit the stage
  if hitTest(sprite(1),point(px,newy)) <> #normal then
    py = newy
  end if

  -- new location for the sprite
  sprite(me.spriteNum).loc = point(px,py)
end
```

一个更复杂的行为可能有一个参数，它的任务是指定该行为所要负责的角色。或者，该行为可以指定一定范围内的角色。

20.5 矢量图形Lingo的故障排除

■ 当采用Lingo对vertexList进行设置时，要确认它是一个有效的顶点的列表，其中所有属性应正确拼写，并且数字的格式应当正确。

■ 如果我们想要在影片播放时用 Lingo改变vertexList而表现动画，应当把centerRegPoint设置为FALSE，把originMode应设置为center，把矢量图形的scaleMode设置为auto-size。否则，当角色的形状改变时，它会在舞台上移动。

■ 如果我们想对矢量图形填充颜色，应当确认 closed属性被设置为TRUE。否则，矢量图形就不会被填充。

■ 矢量图形的顶点越多，显示起来就越慢。无锯齿矢量图形比一般矢量图形的显示要慢。

■ 我们通常需要把一个矢量图形角色的油墨设置为 Background Transparent。如果把它设置为Copy，将导致演员的背景色将被应用到舞台上的整个角色矩形中。

20.6 你知道吗

■ 尽管每个矢量图形演员只能有一条连续的线，但我们可以通过把几个矢量图形重叠放

置的方法来创建更复杂的、类似Freehand或Illustrator的图形。

■ EPS文件实际上就是由顶点和控制柄构成的列表。如果我们很熟悉 EPS文件格式，我们可以写一个可以读此文件并基于其数据创建矢量图形演员的 Lingo程序。

■ 当矢量图形和Flash演员被放置在角色中时，可以像位图一样被旋转、扭曲和缩放。然而，由于它们是由曲线构成的，因而在放大后分辨率也不会降低。

我们可以在消息窗口中使用 Lingo 命令put showProps(member x)，其中x是一个矢量图形演员。该命令能够得到一个有关其全部属性的列表。对于 Flash演员也是一样的。