第30章 声 音 软 件

在多媒体作品里，声音是被忽视的素材。到处都使用着图像和文字，视频文件和影片出尽了风头。但是，用声音作为辅助素材或中心焦点，却能制作出更有趣的多媒体作品。

30.1 钢琴键盘

对于熟悉计算机的声音软件的人来说，制作乐曲是非常容易的。实际上，我们只需录下钢琴或其他乐器的声音，然后把那个声音附加到 Director琴键就可以了。

制作单音调的键盘程序是很容易的，但制作更多的音调，足够用户用来演奏，却比较难。

首先，我们需要制作声音文件本身。 Macromedia的SoundEdit16可以帮助我们轻松完成这件事情。也可以使用其他软件，但用 SoundEdit16更容易些。

第一步要准备一些声音样本，如果可能，就用中央 C调。可以用麦克风和某个乐器，也可以把电子乐器直接插入计算机，或使用音响效果 CD里的声音样本。把这个声音样本做得尽可能地小，并把它存为C3。C指音调， 3指钢琴的一个八度音。我们也可以随意命名，但这种命名结构的效果是很好的。

提示 在制作键盘时，把声音文件做得尽可能地小是很重要的。在一个声音文件里，上下相差5K似乎并不多。但是，如果我们将要制作20个音调，5K就变成了100K。如果把这些音调放在Shockwave短程序里，使用28.8 Modem的用户需要用一分多钟的时间来下载它。

做好C3文件后，可以用它来制作其他音调。按组合键 Command+A，选择全部声音，再从菜单里选择Effect、Pitch Shift。屏幕上会出现如图30-1所示的带有一个小键盘的对话框。

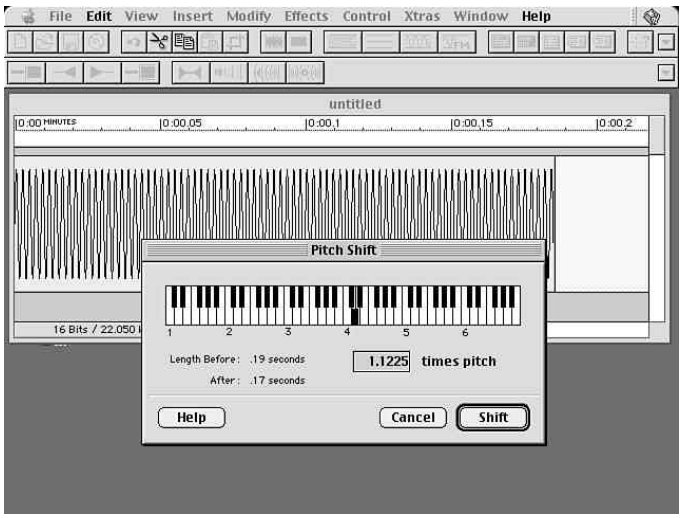

图30-1 SoundEdit16的PitchShift功能可以帮我们用一个音调制作出整个键盘

只要选择想要转换的音调 (如比中央C高一个音的D)，再点击Shift按钮就可以了。声音被转成了该音调。接着把它存储为一个新文件，如 D3。

可以用这种方法继续制作其他音调，但每次都应该用原始的 C3文件制作。用#字符表示黑色键，如C3#。

声音的数量取决于我们想要让钢琴上有多少个琴键。图 30-2的程序里有20个琴键：12个白色的和8个黑色的。它从中央C起，向上跨了一个半八度，到达高音 G。

现在我们已经有了全部声音，可以把它们输入 Director。演员的名称即为文件的名称。图30-3是这部影片的完整的演员表。

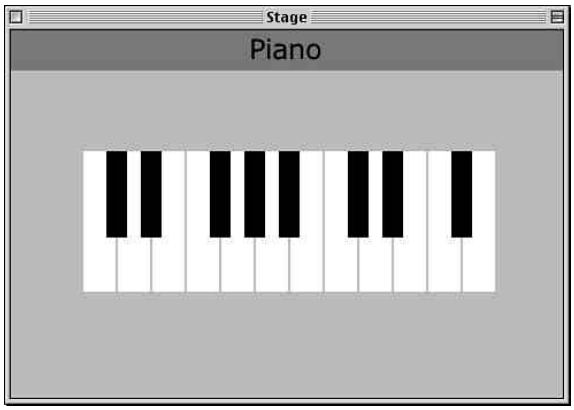

图30-2 一个简单的键盘，其中的琴键是白色和黑色的矩形图形

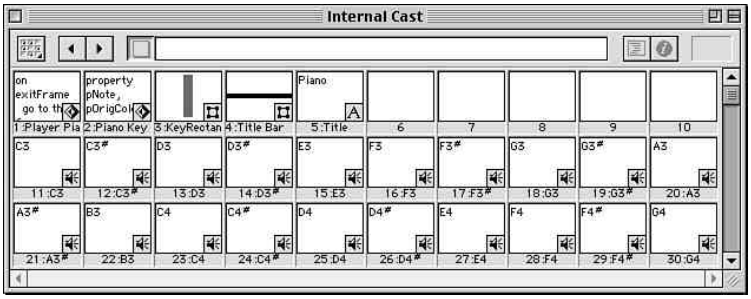

图30-3 演员表窗口显示了键盘所需要的全部声音

要创建图30-2所示的简单的键盘，所需要只是一个图形演员。该演员用于所有角色。有些角色是白色的，有些是黑色的。由于黑色键应该在白色键的前面，因此它们应位于编号较高的通道。

行为可以把这些矩形变为琴键。下面的行为使用了一个参数，以决定哪个声音属于哪个键。它对琴键被按下的另一个反应是把琴键变为红色。

```
property pNote, pOrigColor

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pNote, ¬
    [#comment: "Sound", #format: #sound, #default: VOID]
  return list
```

```
end

on beginSprite me
  -- remember original color
  pOrigColor = sprite(me.spriteNum).color
end

on mouseDown me
  press(me)
end

on mouseEnter me
  -- consider it a press when mouse button is down
  -- and the user rolls over to the key
  if the mouseDown then
    press(me)
  end if
end

on mouseUp me
  unpress(me)
end

on mouseUpOutside me
  unpress(me)
end

on mouseLeave me
  unpress(me)
end

on press me
  -- make key red
  sprite(me.spriteNum).color = rgb(255,0,0)
  -- play note
  puppetSound 1, pNote
end

on unpress me
  -- restore key to black and white
  sprite(me.spriteNum).color = pOrigColor
end
```

这个行为也顾及到了普通琴键所没有的特殊行为。如果用户在某个键以外按下鼠标，又滑向该键，该音符也将演奏。这样就可以只按下一次，随即滑过很多键，弹奏它们。

这个键盘的效果还可以，但还需要一些图像方面的改进。你可以用某个图像处理软件进一步处理琴键，可以为每个键设置被按下时的状态，甚至可以把琴键放成某种角度，制作3D效果。

参见第17章“控制声音”里的“使用 Lingo的声音命令”，可以获得有关使用声音的更多信息。

30.2 演奏的钢琴

另一种钢琴程序是让钢琴自己演奏。我们可以提供一系列音符和音符的持续时间，并在

恰当的时候把它们送给琴键行为。

第一步是要有一个简单的乐谱。用演员的名称指定音符是一个良好的开端，但我们还需要指定音符的持续时间。以1/10秒为单位应该是比较好的方案。

请看下面这个例子：

```
F3.3 0.1 G3#.3 0.1 C4.3
```

这个乐谱用空格把音符分开。每个音符包含一个声音演员名称和一个持续时间，二者用一个“.”分开。0表示休止符。

可以把这样的乐谱存储在纯文本域里。给它取名为 Note List或其他类似的名称。

要演奏这些音符，可以使用帧剧本行为。该行为需要监控乐谱、当前音符、演奏下一个音符的时间和正在演奏的琴键角色 。

```
property pNoteList -- a text list of notes and durations
property pNoteNumber -- the position of the current note
property pEndNoteTicks -- when the current note is done
property pNoteSprite -- the sprite with the current note's key
```

开始，行为要从域里取出乐谱。它始终把乐谱处理为字符串，因为转换为列表并没有什么好处。接着，就从头开始演奏乐曲。

```
on beginSprite me
  -- get the list of notes
  pNoteList = member("Note List").text

  -- start at the first note
  pNoteNumber = 1
  pEndNoteTicks = the ticks
end
```

在每帧结束时，行为都将查看 pEndNoteTicks。这里当前演奏的音符结束的时候，接下去将要演奏另一个音符。

此时，它通过发给当前琴键一个 #unpress消息，关闭当前的琴键。这将调用一个与前面的例子里的处理程序十分相似的 on unpress处理程序。

```
on exitFrame me
  -- ready for next note?
  if the ticks >= pEndNoteTicks then

    -- if there was a previous note, end it
    if not voidP(pNoteSprite) then
      sendSprite(pNoteSprite, #unpress)
    end if

    -- play new note
    playNote(me)
  end if

  go to the frame
end
```

on playNote处理程序找到乐谱里的音符，并演奏它，点前面的部分是声音演员，点后面的部分是持续时间。然后它用持续时间去设置新的 pEndNoteTicks。

然后，它寻找带有琴键行为的角色。在这里，是角色 5~24。当它找到了代表正确的琴键

的角色后，就发出一个#press消息。

```
on playNote me
  -- check to see if done
  if pNoteNumber > pNoteList.word.count then exit

  -- get new note and duration
  noteWord = pNoteList.word[pNoteNumber]
  the itemDelimiter = "."
  note = noteWord.item[1]
  duration = noteWord.item[2]

  -- set end time for note
  pEndNoteTicks = the ticks + 8*value(duration)

  -- if a "0", then just silence, else play note
  if note <> "0" then
    -- find the piano key sprite
    repeat with i = 5 to 24
      if sprite(i).pNote.name = note then
        pNoteSprite = i
        exit repeat
      end if
    end repeat

    -- play it
    sendSprite(pNoteSprite, #press)
  end if

  -- next note
  pNoteNumber = pNoteNumber + 1
end
```

现在，这个行为可以处理任意长的乐谱了。问题难在把我们喜爱的歌曲转换成这些乐谱。CD-ROM里的影片实例里有一首较长的歌，希望能对你有所启发。

我们还可以进一步加工这部影片。可以让两声部同时演奏，一个在声音通道 1，一个在声音通道2；甚至还可以使用非琴键音符，可以直接演奏这些音符，而无须向琴键角色发送消息。

30.3 动态立体声

在不久以前，大多数计算机只有一个小的、低档的音箱，藏在机箱内；如今，即使是最便宜的计算机，也有声卡和一组音箱。

但是，立体声却未被充分利用。作为一个 Director创作者，我们可以轻易地发挥它的作用。

例如，舞台上有一个噪声很大的元素，比如直升飞机。它显然应该发出一架直升飞机的声音。但是，如果用户把直升飞机从舞台的左侧移到右侧，声音却不能从左音箱跑到右音箱。

由于在Director里可以播放多重声音，并能控制每种声音的音量，我们就好象应该能控制每个立体声通道内的音量；但是不可以，至少不能直接实现。

解决的方法是制作两个独立的声音，二者都是立体声。但是，第一个声音是只在左声道里有直升飞机的声音，第二个只在右声道里有直升飞机的声音。图 30-4是出现在SoundEdit 16

编辑窗口里的左声道声音。

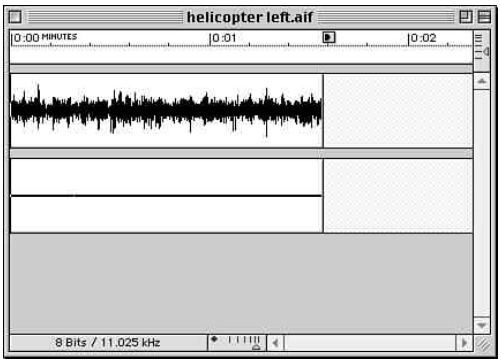

图30-4 声音编辑窗口里是带有左、右两个声道的立体声。但是，只有左声道里有声音，另一个声道里是空的

当直升飞机出现在舞台上时， puppetSound命令让两个声音开始播放。随后，两个声音通道的volume属性由角色在舞台上的横向坐标来调节。

```
on beginSprite me
  -- start sounds
  puppetSound 3, "helicopter left"
  puppetSound 4, "helicopter right"
  setSoundVolumes(me)
end

on setSoundVolumes me

  -- calculate horizontal position
  x = sprite(me.spriteNum).locH
  percent = float(x)/(the stage).rect.width

  -- calculate volumes
  leftVolume = (1.0-percent)*255
  rightVolume = (percent)*255

  -- set volumes
  the volume of sound 3 = leftVolume
  the volume of sound 4 = rightVolume
end


on exitFrame me

  -- sprite follows the cursor
  sprite(me.spriteNum).loc = the mouseLoc

  -- change volumes
  setSoundVolumes(me)
end
```

注释 请注意，这里使用了旧的the句法来设置声音通道，因为sound(3).volume在Director 7中似乎不能用。在更新的Director版本里也许也以试着用一下。

应该把声音设为循环播放。由于声音是由角色触发的，并且受角色的行为控制，我们甚至可以把这样的行为同时赋给多个角色。这个角色使用的是通道 3和通道4，其他角色可以使用其他通道。

30.4 3D声音

真正的3D声音需要环绕音响或相似的设置。但是，通过调节声音通道的音量，我们可以轻易地模仿这种效果。

这个例子不是象前面那个例子那样使用两个声音通道，而是只使用一个声音通道。光标越靠近一个对象，声音就越大，越远离声音就越小。

这个行为相对比较简单。它的volume(音量)的范围是0~255。它计算角色与光标间的距离。当距离大于255时，音量设为零。如果距离较近，则把其值反转，即距离为 0时音量最大，为255，距离为254时音量为1。

```
on beginSprite me
  -- start sounds
  puppetSound 3, "static"
  setSoundVolume(me)
end

on setSoundVolume me

  -- calculate distance
  d = distance(me,sprite(me.spriteNum).loc,the mouseLoc)

  -- calculate volume
  vol = 255-d
  if d < 0 then d = 0

  -- set volumes
  the volume of sound 3 = vol
end

on exitFrame me
  -- change volumes
  setSoundVolume(me)
end

on distance me, p1, p2
  return sqrt(power(p1.locH-p2.locH,2)+power(p1.locV-p2.locV,2))
end
```

尽管CD-ROM里的例子使用的是电视机和干扰噪声，你还可以找到使用这个行为的更巧妙的方法。例如，可以把几个会讲话的对象放在舞台上，当鼠标靠近其中之一时，它的声音就变大了。

这好像我们在博物馆里遇到的情形一样，在那里，每个展区都在播出一些解说词。当我们靠近某个展区，它的声音变得清晰了，而前一个展区的声音则渐渐变弱。

30.5 调节音量控制

如果我们的Director影片有声音效果，应该给用户一个调节音量的机会。的确，每个操作

系统都有一个声音或音量控制面板，但我们可以假设大多数人并不知道它的存在，而且也没有理由要让用户离开我们的程序去调节音量。

添加音量控制并不困难。用系统属性 SoundLevel，可以把音量设置为 0到7。这将调节总体音量，与我们设置的其他volume属性相互独立。它还可以控制系统的警告声音。

30.5.1 音量滑动条

添加音量控制的第一步是确定想要什么样的控制方式。常用的是滑动条。图 30-5里有一个滑动条控件，此外还有单个按钮控件，在本章的后面将要讲到它。

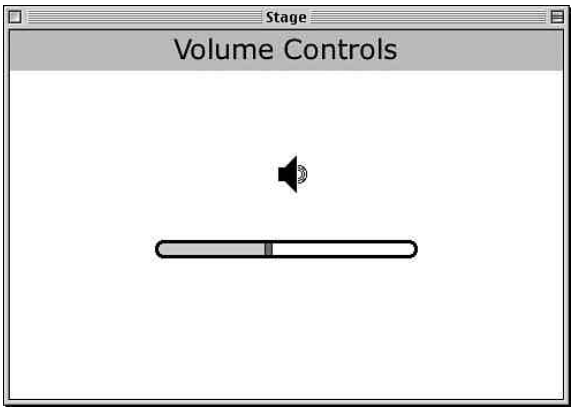

图30-5 图中有两种不同的音量控件：滑动条和单个按钮控件

滑动条行为与第15章“图形界面元素”里的滑动条行为非常相似，其中包含一个滑动条、一个阴影和一个背景图形。行为是附属滑动条的，它让阴影角色跟随它移动。

在这个行为里有许多内容都与那个滑动条行为里的内容相似，只是有些参数不见了。音量的最大和最小值总是0和7，因此没有必要把它们设成参数。这个行为里也没有初始值参数，因为它取的是计算机当前被设定的音量级别。

事实上，根本没有必要存储滑动条的当前值。当用户移动滑动条时，它设置了 soundLevel 系统属性，这将代替pValue属性。

```
property pPressed -- whether the sprite is being pressed
property pBounds -- the rect of the shadow sprite at start
property pShadowSprite -- the number of the shadow sprite
property pMinimumValue, pMaximumValue -- use by the marker sprite only

on getPropertyDescriptionList me
  list = [:]
  addProp list, #pShadowSprite, [#comment: "Shadow Sprite",¬
    #format: #integer, #default: 0]
  return list
end

on beginSprite me
  pBounds = sprite(pShadowSprite).rect
  pMinimumValue = 0
  pMaximumValue = 7
```

```
  setMarker(me)
  setShadow(me)
end

on mouseDown me
  pPressed = TRUE
end

on mouseUp me
  pPressed = FALSE
end

on mouseUpOutside me
  pPressed = FALSE
end

on exitFrame me
  if pPressed then
    moveMarker(me)
  end if
  setMarker(me)
  setShadow(me)
end

-- this handler takes the mouse position and figures the
-- value of the slider
on moveMarker me
  -- compute the position as a number between 0 and 1
  x = the mouseH - pBounds.left
  sliderRange = pBounds.right-pBounds.left
  pos = float(x)/sliderRange

  -- translate to a value
  valueRange = pMaximumValue - pMinimumValue
  val = pos*valueRange + pMinimumValue
  val = integer(val)

  -- check to make sure it is within bounds
  if val > pMaximumValue then
    val = pMaximumValue
  else if val < pMinimumValue then
    val = pMinimumValue
  end if

  the soundLevel = val
end


-- this sets the marker sprite
on setMarker me
  -- compute the value as a number between 0 and 1
  valueRange = pMaximumValue - pMinimumValue
  sliderPos = float(the soundLevel)/float(valueRange)

  -- translate to a screen position
```

```
  sliderRange = pBounds.right-pBounds.left
  x = sliderPos*sliderRange + pBounds.left

  -- set marker
  sprite(me.spriteNum).locH = x
end

-- this handler lets the marker sprite set the shadow sprite
on setShadow me
  x = (sprite me.spriteNum).locH
  r = rect(pBounds.left, pBounds.top, x, pBounds.bottom)
  sprite(pShadowSprite).rect = r
end
```

请注意，这个行为里的on exitFrame处理程序设置了音量，而不在乎滑动条是否正被移动。这表明滑动条应对the soundLevel的变化有所反应，尽管滑动条并不影响那些变化。对此我们可以进行测试，方法是播放影片，并用消息窗口设置 the soundLevel。滑动条对发生的变化将做出反应。当用户决定用计算机的控制面板手工调节音量时，情况也是这样。这保证滑动条总是显示当前的声音设置。

参见第15章里的15.5节“ 创建滑动条 ”，可以获得更多有关创建滑动条的信息。

30.5.2 音量按钮

尽管音量滑动条的效果不错，但有时却难于付诸实施，也许因为屏幕上没有足够的位置，也许因为滑动条与我们的界面设计不和谐。

我喜欢用一种更简单的方法。把一个按钮放在舞台上，用户按它，就可以改变音量。每按一次，音量就上升一级。如果音量到达最大，再按一次，就变小。

这个按钮可以用8个图形代表the soundLevel的8个级别。在图 30-5的滑动条上方的就是这样一个按钮。图 30-6的演员表里包含这个按钮的8种情形。

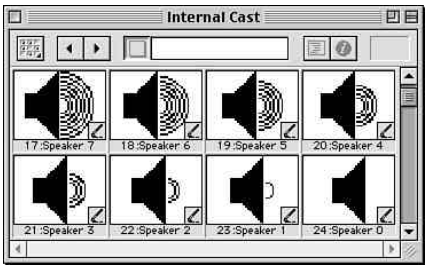

图30-6 演员表里的8个图形代表音量按钮的8个级别

控制这个按钮的行为比滑动条的行为简单得多。但同滑动条的行为一样，它在每一帧都设置按钮的演员，而不管用户有没有按按钮。这保证即使当用户手工调节音量时，按钮能够正确地反应出当前的音量级别。

事实上，我们可以把两种控制音量的工具同时放在舞台上，如图 30-5所示。当我们调节其中之一时，另一个也会有所反应。也可把按钮与音量联系在一起，而滑动条只可以体现声音的变化，却不能被点击。

```
on beginSprite me
  setIcon(me)
end

on setIcon me
  val = the soundLevel
  sprite(me.spriteNum).member = member("speaker"&&val)
end
```

```
on mouseDown me
  val = the soundLevel
  if val < 7 then
    val = val + 1
  else
    val = 0
  end if
  the soundLevel = val
  setIcon(me)
end

on exitFrame me
  setIcon(me)
end
```

可以用任何8个演员表示8种音量。用尺寸不同的扩音器，或使用从耳语到呼喊的人的图像，都可以得到很具创意的效果。

30.6 声音软件的故障排除

■ 如果你制作的是多通道的混声，在播放声音时是否出现严重的滞后现象？应该检查影片是否运行在带有the platform的Windows计算机上。如果是，试着把the soundDevice调节为QT3Mix。如果用户安装了QuickTime 3，这将得到很好的混声效果；如果没有安装，the soundDevice转换为MacroMix。

■ 有时，Windows在支持多通道声音播放时会有问题。因此创作者应该经常在用户使用的那种档次的计算机上进行测试，并在说明书里写明系统配置要求。Macromedia的Tech Support Web网站为声音故障的排除提供了重要的信息。

■ 如果你制作的是立体声，但当生成Shockwave影片后，却变成了单声道声音，应该检查Xtras菜单里的Shockwave Audio Settings。有时，为了减小影片的数据量，这里会设置为把所有立体声都转换为单声道声音。

30.7 你知道吗

■ 如果你不喜欢滑动条或单个按钮的音量控制法，你还可以有多种选择。可以使用第16章“控制文本”里的那类行为，制作基于文本的弹出菜单，也可以使用一组单选按钮。也可以使用拨号方式，用8个图形代表8种音量，当用户点击右边的拨号钮时，音量加大，点击左边的拨号钮时，音量减小。

■ 可以使用第15章里的复选框行为中的一些程序代码制作一个静音按钮。只要让该行为记住当前音量，并在on位置继续就可以了。如果用户点击该复选框，the soundLevel将被设为0；当他再次点击时，把the soundLevel重设为它原来的值。或者，用种更好的方法，使用the soundEnable属性。这样，我们可以不修改the soundLevel就关闭声音。

■ 可以用声音的volume属性使声音逐渐消失或逐渐出现，但Lingo的sound fadeIn和sound fadeOut命令正可以实现这种效果。把两个通道同时播放，可以实现由一个通道渐渐变化至另一个通道。

■ 把声音放在QuickTime 3声道，使我们能更好地控制播放哪种声音，停止哪种声音。可以同步地使用多声道，并用setTrackEnable分别设置每个声道的开和关状态。