



Materialerstellung: Grafik, Text und Animation

1

Die Basis jeder Director-Produktion sind gut vorbereitete Ausgangsmaterialien – Texte, Bildschirmgestaltung, Buttons, Sounds, Videos, eventuell auch vorproduzierte Animationen und ähnliches – und detaillierte Anweisungen, wie diese in der Anwendung zu verwenden sind. Üblicherweise sind eine ganze Reihe von Spezialisten am Werk, die Texte produzieren, Grafiken erstellen oder in anderer Weise zur Produktion beitragen. Und üblicherweise entsteht so ein beträchtlicher Abstimmungsbedarf – oder es ist in Ihrem Interesse, daß er entsteht, denn wenn eine detaillierte Abstimmung fehlt, ist das gesamte Produktionsteam und insbesondere der Director-Programmierer der Leidtragende. Den letzten beißen die Hunde, das gilt erst recht bei Multimedia-Produktionen.

Director bietet bei Grafik, Text und Animation besonders zahlreiche Optionen, wie – intern oder extern – Medien erstellt, weiterbearbeitet, korrigiert oder animiert werden können. Ein effektiver Produktionsablauf kann je nach Projekt ganz unterschiedlich aussehen – den einen richtigen Ansatz wird es nicht geben. Vielmehr wird der ideale Mix von Erstellungs- und Bearbeitungsschritten außerhalb des Authoring-Programmes, den geeigneten Austauschformaten und der Bearbeitung in Director ganz unterschiedlich aussehen können.

In diesem Kapitel werden Sie daher auch alte Bekannte wiedertreffen: Selbst das vielgeschmähte Malfenster bietet Features, die auch erfahrenen Director-Anwendern nicht bekannt sind. Sie werden aber vor allem Hinweise finden, wie Sie – innerhalb oder außerhalb von Director – Arbeitsabläufe effektiv gestalten können. Ich gehe dabei davon aus, daß der Director-Programmierer häufig weniger mit gestalterischen Fragen oder mit der Texterstellung zu tun hat, daß er aber in bezug auf Grafiker, Texter etc. zwei wesentliche Aufgaben hat:

- Er definiert die „Schnittstelle“ (Grafik- und Textformate, Benennung, „Binnenstruktur“ von Photoshop- oder Textdateien etc.) so, daß die an das Produktionsteam gestellte Aufgabe
 - a) überhaupt realisierbar ist und
 - b) mit möglichst wenig Aufwand realisierbar ist.

Dieses Kapitel gibt Ihnen einen Überblick über die zahlreichen Optionen zur Erstellung des Ausgangsmaterials für eine Director-Produktion

Selbstverständlich haben Sie in Photoshop oder einem anderen Bildbearbeitungsprogramm bessere Möglichkeiten, weiche Auswahlen einzusetzen. Ob die hier vorgestellte Methode ihren Zweck erfüllt, hängt vor allem von Ihren Ansprüchen ab – für schnelle Korrekturen ist sie allemal brauchbar.

Vectorshapes erstellen und importieren

Eine Eigenschaft von Vectorshape-Darstellern unterscheidet sie von allen anderen grafischen Darstellertypen: Sie sind die einzigen Darsteller, deren Inhalt zur Laufzeit – also auch in Shockwave und im Projektor – vollständig manipuliert werden kann. Alle anderen Grafikdarsteller können zwar als Sprites animiert werden – ihre Inhalte aber bleiben unverändert. Dieser neue Darstellertyp erfordert daher an einigen Stellen ein Umdenken. Jede seiner `member properties` ist direkt durch Lingo beeinflussbar, sei es die Füllfarbe oder die Krümmung eines Kurvenverlaufs an einem bestimmten Punkt. Daneben können Vectorshapes natürlich auch wie andere Darstellertypen im Director-Drehbuch kombiniert und animiert werden.

In ihrer „Außenansicht“ ähneln Vectorshapes dem Flash-Darstellertyp. Die gleiche Program-Engine visualisiert (oder „rendert“) Flash- und Vectorshapes auf der Bühne. Über die gemeinsamen Darstellungs- und Animationsoptionen der beiden Formate finden Sie daher in Kapitel 2 mehr.

Wenn Ihnen die einschlägigen Vektorzeichenprogramme (Adobe Illustrator, Macromedia Freehand, Corel Draw etc.) bekannt sind, werden Sie auf Anhieb mit der Erstellung von Vectorshape-Darstellern in Director zurechtkommen. Sie können mit Director – im Unterschied zu den genannten Programmen – nur einfache, aus einer einzigen Linie bestehende Vektordarsteller erstellen. Komplexere Vektorgrafiken sollten Sie außerhalb von Director produzieren und als Flash-Darsteller in Director benutzen.

Übrigens werden Sie einen Menübefehl zum Import von EPS-Dateien als Vectorshapes vergeblich suchen. Auch über die Zwischenablage lassen sich Illustrator- oder Freehand-Kurven nicht in Director integrieren. Tatsächlich hat Macromedia keine Importfunktion vorgesehen – was nicht so schlimm ist, da mit Darrel Plant ein Director-Anwender ein entsprechendes Importskript für Illustrator-EPSe erstellt hat. Eine Kopie davon befindet sich auf der beiliegenden CD-ROM; aktuellere Versionen gibt es möglicherweise auf der Website von Darrel Plant unter <http://www.moshplant.com/director/ps2vs/>. Das Skript ist eine überaus spannende Lektüre: Es nimmt die (Text-)Informationen der EPS-Datei, interpretiert sie zeilenweise und übersetzt sie in Darsteller-Properties von Vectorshapes. Allerdings gilt die gleiche Einschränkung wie für Vectorshapes, die im Director-eigenen Editor erstellt werden: Ein Dar-



► 249

Das Importskript für Illustrator EPSe finden Sie auf der CD-ROM im Verzeichnis WORKSHOP\KAPITEL01\EPS

Der Vectorshape-Editor enthält auch ein süßes „Osterei“ (d.h. ein von den Programmierern eingefügtes, verborgenes Feature): Tippen Sie „ABC“ im Editorfenster – danach werden alle Eingaben in einfache Vektorbuchstaben umgesetzt

steller kann nur eine einzige Vektorlinie enthalten. Sind in der EPS-Datei mehrere Linien enthalten, werden von dem Skript auch mehrere Darsteller angelegt.

Das Zeichenwerkzeug des Vectorshape-Editors dient zum Erstellen von Bezierkurven und zum Hinzufügen von Punkten zu einer bestehenden Kurve. Einfaches Klicken fügt dabei einen Eckpunkt ein, Klicken und Ziehen einen Kurvenpunkt, dessen beide Vertex-Anfasser gekoppelt sind. Um den Kurvenverlauf an einem Eckpunkt nachträglich anzupassen, nutzen Sie das Pfeilwerkzeug bei gedrückter $\text{⌘}/\text{Alt}$ -Taste.

Geschlossene Vectorshapes können Sie mit dem Zeichenwerkzeug erstellen, indem Sie den letzten Punkt direkt auf den ersten plazieren – Sie erhalten in der Anzeige dann eine entsprechende Rückmeldung. Außerdem dient das Ankreuzfeld „geschlossen“ im Editor diesem Zweck. Die Optionen oberhalb der Arbeitsfläche des Editors dienen der genauen Festlegung von Verläufen – besonders interessant ist die Möglichkeit, den x- und y-Wert des Ausgangspunktes von Verläufen zu verschieben (wie in Abbildung 1.15 geschehen), oder den Winkel von linearen Verläufen anzugeben. Mit den Optionen VERTEILUNG und DURCHLÄUFE können Sie – ähnlich wie im Malfenster – die Farbverteilung innerhalb des Verlaufes und die Anzahl der Farbdurchläufe bestimmen.

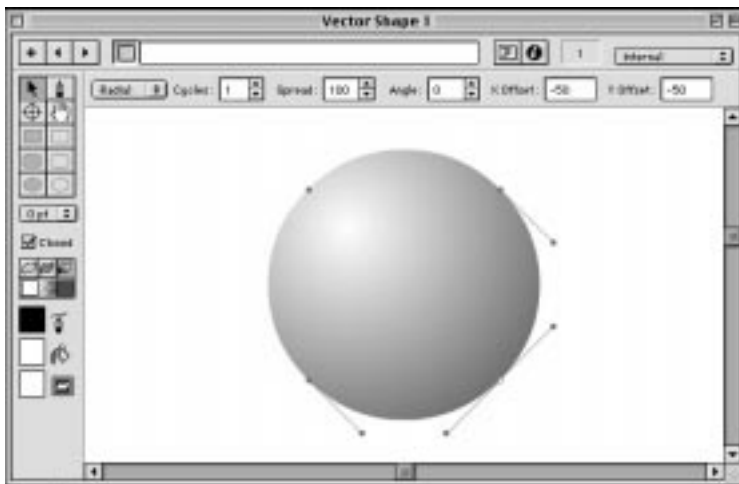


Abbildung 1.14:
Der Editor für Vectorshapes:
Werkzeuge, Farbwahl und Eingabefelder für Verlaufsoptionen

Die Besonderheit der hier erstellten Grafiken ist, daß sie vollständig in Directors Scripting-Sprache Lingo beschreibbar sind. Der Kreis mit Verlaufsfüllung läßt sich mit folgenden Parametern beschreiben (eine vollständige Auflistung aller Properties erhalten Sie mit dem Lingo-Befehl `member(m).showProps()`, den Sie im Nachrichtenfenster eingeben können).

```
-- Registrierungspunkt; zentriert?
regPoint:          point(140, 140)
centerRegPoint:    1
-- Originalgröße
defaultRect:       rect(0, 0, 281, 281)
-- Kurve geschlossen?
closed:            1
-- Linienstärke
strokeWidth:       0.0000
-- Füllungstyp
-- (Verlauf: #gradient, Farbe: #solid, keine: #none)
fillMode:          #gradient
-- Art des Verlaufs: #radial oder #linear
gradientType:      #radial
-- Farbverteilung im Verlauf
fillScale:         100.0000
-- Richtung (für lineare Verläufe)
fillDirection:    0.0000
-- Startpunkt des Verlaufs (Versatz x und y)
fillOffset:        point(-50, -50)
-- Anzahl der Durchläufe des Verlaufs
fillCycles:        1
-- Farbe der Linie, der Füllung, des Hintergrunds und
-- Endfarbe des Verlaufs
strokeColor:       rgb( 0, 0, 0 )
fillColor:         rgb( 221, 221, 221 )
backgroundColor:   rgb( 255, 255, 255 )
endColor:          rgb( 255, 0, 0 )
-- Beschreibung der Kurve mit Punkten und Position der
-- Vertex-Anfasser
vertexList:        [[#vertex: point(-98.0000, -98.0000),
#handle1: point(55.1600, -55.1600), #handle2: point
(-55.1600, 55.1600)], [#vertex: point(99.0000, -98.0000),
#handle1: point(55.1600, 55.1600), #handle2: point
(-55.1600, -55.1600)], [#vertex: point(99.0000, 99.0000),
#handle1: point(-55.1600, 55.1600), #handle2: point
(55.1600, -55.1600)], [#vertex: point(-98.0000, 99.0000),
#handle1: point(-55.1600, -55.1600), #handle2: point
(55.1600, 55.1600)]]
```

Ebenso wie Sie im Vectorshape-Editor eine Grafik ändern, können Sie mit Lingo-Befehlen einzelne Aspekte der Grafik manipulieren. Folgende Statements ändern beispielsweise den Verlauf in unserem Kreis aus Abbildung 1.14 in einen linearen, senkrechten Verlauf mit leichtem Versatz:

```
member(1).gradientType = #linear
member(1).fillDirection = 90.0
member(1).fillOffset = point(0,20)
```

Auch einzelne Vertexpunkte können Sie skriptgesteuert hinzufügen (addVertex), entfernen (deleteVertex), bewegen (moveVertex) und die von ihnen bestimmte Kurvenform ändern (moveVertexHandle).

Arbeitsbeispiel

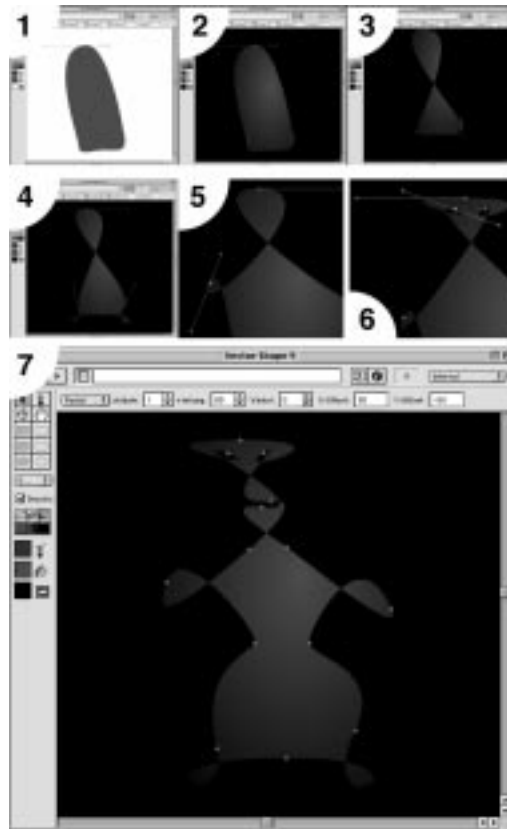
Arbeitsbeispiel Vectorshape-Editor

Komplexere Formen – wie beispielsweise den Gnom aus Abbildung 1.15 – lassen sich im Vectorshape-Editor am besten aus einfachen Flächen heraus entwickeln.

Schritt 1:

Erstellen Sie eine einfache, gefüllte Fläche mit 4 Punkten (klicken und ziehen, damit die Anfassers ausgezogen werden) und geben Sie die Füllungsfarbe an. Sie können auch einen Kreis als Ausgangsform nehmen. Setzen Sie die Linienstärke auf 0. Falls die Form nicht gefüllt ist, klicken Sie auf das Ankreuzfeld „geschlossen“ und auf das Icon für „gefüllt“.

Abbildung 1.15:
Die Arbeitsschritte zur Gnomerstellung
im Vectorshape-Editor



Schritt 2:

Legen Sie Füllungsfarbe und Endfarbe des Verlaufs sowie – mit den Werkzeugen in der oberen Leiste – einen eventuellen Versatz in x- und y-Richtung fest.

Schritt 3 und 4:

Trennen Sie jeweils einen Teil der Fläche ab, indem Sie einen der Anfasser des oberen Punktes um seine eigene Achse drehen (Pfeilwerkzeug). Ohne weitere Punkte einzufügen, können Sie Kopf und Füße so erstellen. Falls nötig, bewegen Sie auch die Punkte mit dem Pfeilwerkzeug in die entsprechende Richtung.

Schritt 5 und 6:

Erstellen Sie weitere Punkte, indem Sie mit der Zeichenfeder auf die Vektorlinie klicken und ohne loszulassen ziehen. Die Aussparungen für die Augen erreichen Sie, indem Sie die Anfasser so verdrehen, daß sich die beiden Flächen überschneiden.

Schritt 7:

Auch der Mund wird mit zusätzlichen Punkten und entsprechender Krümmung aus der Fläche modelliert. Die Taille ist ein Eckpunkt: Fügen Sie einen Punkt ein, ohne zu ziehen, und korrigieren Sie die Anfasser mit dem Pfeilwerkzeug bei gedrückter `[Alt]`-Taste.

Die Vektorfigur können wir auch tanzen lassen. Eine einfache Animationsmethode könnte so aussehen, daß Sie die so geschaffene Vektorform duplizieren (BEARBEITEN / DUPLIZIEREN), einige Vektorpunkte bewegen, wiederum duplizieren und verzerren, bis Sie die gewünschten Tanzbewegungen erstellt haben. Wählen Sie dann alle so erstellten Darsteller im Besetzungsfenster aus und fügen Sie sie mit MODIFIZIEREN / DARSTELLER IN KANAL ins Drehbuch ein.

Eine andere Methode nutzt Lingo, um Parameter des Vectorshapes zu modifizieren. Statt die Darsteller zu duplizieren, können Sie z.B. auch nur die Vertexlist, die Beschreibung der Vektorform, speichern:

Folgendes erhalten Sie im Nachrichtenfenster:

```
put member("gnom").vertexlist  
  
[[#vertex: point(-51, -234), #handle1: point(-173, 0),  
#handle2: point(172, 0)], [#vertex: point(-67, -216),  
#handle1: point(-83, -30), #handle2: point(70, 25)],  
[#vertex: point(-22, -144), #handle1: point(-65, -7),  
#handle2: point(99, 11)], [#vertex: point(12, -90),  
#handle1: point(14, 11), #handle2: point(-14, -11)],  
[#vertex: point(152, -7), #handle1: point(-45, -101),  
#handle2: point(22, 49)], [#vertex: point(42, 39),  
#handle1: point(0, 20), #handle2: point(0, -20)],  
[#vertex: point(104, 157), #handle1: point(-67, 144),  
#handle2: point(40, -86)], [#vertex: point(11, 194),
```

Ein Behavior ist ein Skript, das einem Sprite auf der Bühne oder einem Frame zugeordnet ist. Ziehen Sie das Sprite ins Drehbuchfenster und legen Sie ein **NEUES VERHALTEN** an:



```
#handle1: point(-241, 0), #handle2: point(241, 0],
[#vertex: point(-82, 181), #handle1: point(-25, -111),
#handle2: point(22, 100)], [#vertex: point(-31, 39),
#handle1: point(5, -16), #handle2: point(-5, 15)],
[#vertex: point(-149, -43), #handle1: point(-38, 97),
#handle2: point(23, -59)], [#vertex: point(-39, -87),
#handle1: point(-17, 17), #handle2: point(17, -17)],
[#vertex: point(-10, -152), #handle1: point(-94, 10),
#handle2: point(60, -6)], [#vertex: point(-19, -217),
#handle1: point(-53, 34), #handle2: point(47, -30)]]
```

Ich habe mir für vier Tanzzustände der Figur jeweils eine solche vertexList ausgegeben lassen und sie in die Felddarsteller „list1“ bis „list4“ kopiert. Außerdem will ich auf der Bühne die Beleuchtung – also den Lichtpunkt der Füllung des Vectorshapes – mit der Maus über die Figur wandern lassen.

Dieses Behavior sorgt für die Animation durch Setzen der vertexList:

```
property counter

on beginsprite me
  counter = 0
end

on exitframe me
  counter = (counter + 1)
  i = (counter mod 4) + 1 -- liefert Werte zw. 1 und 4
  -- Einlesen der Vertex-Listen aus Felddarstellern
  myvertexlist = member("list"&i).text.value
  -- Setzen der Vertexlist des Darstellers
  member("gnom").vertexlist = myvertexlist
end
```

Ergänzt wird es durch einen mouseWithin-Handler, der den Startpunkt des kreisförmigen Verlaufs der Füllung in Abhängigkeit von der Mausposition neu setzt, solange sich die Maus über der Figur befindet:

```
on mousewithin me
  mmb.fillOffset = stageToFlash (sprite(me.spritenum), ↵
the mouseLoc) - point (sprite(me.spritenum).width/2, ↵
sprite(me.spritenum).height/2)
end
```

Bei Klick soll eine weitere Aktion erfolgen: Für einen kurzen visuellen Effekt wird der kreisförmige Verlauf vervielfacht. Maximal kann die Füllung sieben Mal wiederholt werden, so daß wir hier nur einfach von 1 nach 7 und wieder auf 1 zählen.

```
on mouseup me
  repeat with cycl in [1,2,3,4,5,6,7,6,5,4,3,2,1]
    member("gnom").fillCycles = cycl
    updatestage
  end repeat
end
```

Es bietet sich an, an diesem Beispiel mit weiteren Darsteller-Properties zu spielen. Damit Sie Ihr Originalbild immer wieder restaurieren können, sollten Sie alle wichtigen Parameter im `beginSprite`-Handler in Properties sichern und einen eigenen `reset`-Handler anlegen.

```
property counter, mc, mb, mec, fs,fo,fc

on beginsprite me
  counter = 0
  mc = member("gnom").fillcolor
  mb = member("gnom").backgroundcolor
  mec = member("gnom").endcolor
  fs = member("gnom").fillscale
  fo = member("gnom").filloffset
  fc = member("gnom").fillcycles
end

on reset me
  member("gnom").backgroundcolor = mb
  member("gnom").fillcolor = mc
  member("gnom").endcolor = mec
  member("gnom").fillscale = fs
  member("gnom").filloffset = fo
  member("gnom").fillcycles = fc
  counter = 0
end
```

Ein solches Reset können Sie z.B. im gleichen Behavior auf `mouseLeave` anlegen. Auf jeden Fall sollten Sie beim Authoring folgenden Eintrag im `stopMovie`-Skript haben, um alle Sprites des aktuellen Frames beim Anhalten des Filmes zurückzusetzen.

```
on stopMovie
  sendAllSprites(#reset)
end
```

Übrigens können Sie Vectorshapes auch als Grundlage für Pfadanimationen nutzen (sogenannte „Motion Guides“). Die entsprechende Animationstechnik wird im Abschnitt „Animation & Lingo“ in diesem Kapitel erläutert.

Das fertige Arbeitsbeispiel finden Sie auf der beiliegenden CD-ROM im Verzeichnis WORKSHOP \ KAPITEL01 \ VEKTOR.

► 177

würde Sprite 27 hinter Sprite 26 plazieren, und zwischen beiden wären noch 49 mögliche Layer für andere Sprites.

Pfadanimationen: Vectorshapes als „Motion Guides“

Für Pfadanimationen stehen im Director-Drehbuch mit Keyframes und Tweening (vgl. Abschnitt „Tweening“) eigentlich ausreichende Werkzeuge zur Verfügung. Wer noch mehr Kontrolle braucht, kann in Director 7 Sprite-Bewegungen auch an die Outline von Vectorshapes koppeln. Das bietet einige Vorteile:

- Die gesamte Animation kann auf einem Frame ablaufen.
- Der Animationspfad ist einfach im Vectorshape-Editor zu modifizieren.
- Es können auch Pfade beschrieben werden, die mit Tweening nicht möglich sind.
- Der Pfad kann einfach bewegt oder skaliert werden. Mit etwas mehr Scripting-Aufwand wäre auch das Rotieren oder anderweitige Verzerren des Pfades möglich.

Das Behavior kann auf ein beliebiges Sprite gezogen werden. Im gleichen Frame muß ein benanntes Vectorshape liegen, das Sie dann als „Motion Guide“ angeben können.

Sie können die Outline des Vectorshapes natürlich auch unsichtbar machen, indem Sie die `strokewidth` des Darstellers auf 0 setzen.

Das Behavior erlaubt es, eine Gesamtzeit für einen Umlauf anzugeben; Änderungen der Frame-Rate wirken sich also nur auf die Genauigkeit, nicht auf die Geschwindigkeit der Bewegung aus.

```
-- MotionGuide-Behavior. Original: Macromedia
-- Bearbeitung: Joachim Gola

property mySprite, myGuideMember, myPeriod
property myGuideSprite, myGuideVerts

on beginSprite me
  mySprite = sprite(me.spriteNum)
  myGuideMember = member myGuideMember
  myPeriod = myPeriod * 60 -- Umwandlung in Ticks
  myGuideSprite = sprite(FindGuideSprite(myGuideMember))
  checkclosed me
end
```



Auf der CD-ROM finden Sie ein Anwendungsbeispiel und das Behavior „Motion Guide“
(WORKSHOP \ KAPITEL01 \ VEK-TOR)

Für geschlossene Pfade wird – falls nicht schon vorhanden – ein Endpunkt eingefügt, der dem Startpunkt entspricht.

```

on checkclosed me
  if myguidemember.closed then
    x = myguidemember.vertexlist.count
    if (myguidemember.vertexlist[x] <> myguidemember.vertexlist[1]) then
      tempvl = myguidemember.vertexlist
      tempvl.append(tempvl[1])
      myguidemember.vertexlist = tempvl
    end if
  end if
end

```

Im prepareFrame-Handler werden alle nötigen Berechnungen für die neue Sprite-Position ausgeführt.

```

on prepareFrame me
  -- Position im Gesamtdurchlauf und Segment relativ zur
  -- Gesamtzeit des Durchlaufes
  rawTime = ((the ticks) mod ↵
integer(myPeriod)) / float(myPeriod)
  numSegments = count(myGuideMember.vertexList) - 1
  whichSegment = integer( .5 + rawTime * numSegments)
  segmentStart = float(whichSegment - 1) / numSegments
  t = (rawTime - segmentStart) * numSegments
  coeffs = GetCoefficients(myGuideMember, whichSegment)
  basePos = ↵
(myGuideMember.vertexList)[whichSegment].vertex
  -- Berechnung der Position in Member-Koordinaten
  -- newPt = f(t)
  newPt = (power(t,3) * coeffs.a) + (power(t,2) * ↵
coeffs.b) + (t * coeffs.c) + basePos
  -- Umrechnung in Bühnenkoordinaten
  mySprite.loc = flashToStage(myguidesprite, newPt) + ↵
myGuideSprite.loc - point(myGuideSprite.left, ↵
myGuideSprite.top)
end

```

Die Umrechnung der Darsteller- in Bühnenkoordinaten mit `flashtostage` bietet die Möglichkeit, auch ein skaliertes Sprite als Motion Guide zu nutzen.

Die Hilfsfunktion `GetCoefficients` liefert die Informationen des vorangehenden und des folgenden Vertex-Punktes des Vectorshapes für die Berechnung der neuen Sprite-Position zurück:

```
on GetCoefficients whatShape, whichSegment
  vList = whatShape.vertexList
  n = whichSegment
  -- Absolute Position:
  if voidP(vList[n].getaprop(#handle1)) then ↵
    vList[n].setaProp(#handle1, 0)
  firstHandleAbs = vList[n].vertex + vList[n].handle1
  if voidP(vList[n+1].getaprop(#handle2)) then ↵
    vList[n+1].setaProp(#handle2, 0)
  secondHandleAbs = vList[n+1].vertex+vList[n+1].handle2
  c = 3 * vList[n].handle1
  b = 3 * (secondHandleAbs - firstHandleAbs) - c
  a = vList[n+1].vertex - vList[n].vertex - c - b
  return [#c: c, #b: b, #a: a]
end
```

Die folgenden Hilfsfunktionen erlauben es, ein Vectorshape-Sprite auf der Bühne aufzuspüren bzw. den aktuellen Motion Guide on-the-fly umzuschalten.

```
on FindGuideSprite whichShape
  repeat with i = 1 to 1000
    if sprite(i).member = whichShape then return i
  end repeat
  alert "no guide member on screen"
  halt
end

on MotionGuide_ChangeGuide me, whatMember
  myGuideMember = whatMember
  myGuideSprite = sprite(FindGuideSprite(myGuideMember))
end
```

Im `getPropertyDescriptionList`-Handler wird der Dialog definiert, der die Konfiguration des Behaviors erlaubt:

```
on getPropertyDescriptionList me
  theProps = [:]
  c = "Name des Motion Guides:"
  r = []
  repeat with i = 1 to 1000
    if sprite(i).member.type = #vectorShape then add r, ↵
      sprite(i).member.name
  end repeat
  addProp theProps, #myGuideMember, ↵
  [#comment:c, #format:#string, #range:r, #default:1]

  c = "Sekunden pro Durchlauf:"
  r = [#min: 0.5, #max: 20]
  addProp theProps, #myPeriod, ↵
  [#comment:c, #format:#float, #range:r, #default: 5]

  return theProps
end
```

A

Darsteller- und Sprite-Properties, darstellerspezifische Funktionen

Die folgende Referenz nennt Ihnen die Darsteller- und Sprite-Properties für die wichtigsten visuellen Darstellertypen (`#digitalVideo`, `#quickTimeMedia`, `#flash`, `#animGIF`, `#vectorshape`, `#bitmap`, `#text`, `#field`). Die für alle Typen verfügbaren Properties werden zu Beginn aufgeführt; sie sind in den Unterkapiteln aber jeweils noch einmal genannt. Alle Unterkapitel führen die Properties nach folgendem Muster auf:

1. Darstellereinhalte/-referenzen
2. Speicher, Streaming
3. Größe, Ort, Sichtbarkeit, Cropping, Rotation
4. Playback, Loop
5. Zeit, Frame-Rate
6. Quality, Blend, Alpha, Antialias, Mask
7. Farbe
8. Interaktivität, Sound, Scripts

Falls weitere Kategorien verfügbar sind, so sind für einzelne Darstellertypen weitere Unterkapitel angehängt.

Wo vorhanden, sind Funktionen, die nur für bestimmte Darstellertypen Gültigkeit haben, ebenfalls aufgeführt.

Für alle Darstellertypen gültig**Darstellerinhalte/-referenzen***(Properties:)*

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
castlibNum	Besetzungsnummer	•		•	Die interne (Standard-)Castlib hat die Nummer 1
fileName	String	•			vollständige Pfadangabe oder URL
memberNum	Darstellernummer	•	•	(•)	für Sprites setzbar
modified	TRUE/FALSE	•		•	
name	String	•			
number	interne Darstellernummer	•		•	inkl. Information über die Castlib
size	Größe in Bytes	•		•	
type	#bitmap, #flash, #alphamania etc.	•	•		
thumbnail	<Objekt>	•			

Speicher, Streaming*(Properties:)*

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
loaded	TRUE/FALSE	•			
mediaReady	TRUE/FALSE	•			
preload	TRUE/FALSE	•			
purgePriority	0-3	•			legt die Entladereihenfolge fest: 3: normal; 0: nie

(Funktionen:)

- preloadNetThing (*url*)
- downloadNetThing *URL, PfadundDateiName*
- preload *member(m) {,member(m)}*

Für alle Darstellertypen gültig

Größe, Ort, Sichtbarkeit, Cropping, Rotation

(Properties:)

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
regpoint	Punkt	•			in Darstellerkoordinaten
height	Anzahl Pixel	•	•		
rect	rect (x1, y1, x2, y2)	•	•		
width	Anzahl Pixel	•	•		
stretch	TRUE/FALSE		•		obsolet
loc	Punkt		•		Position des RegPoints in Bühnenkoordinaten
locH	Anzahl Pixel		•		ebenso, horizontal
locV	Anzahl Pixel		•		ebenso, vertikal
locZ	-2.000.000.000 – 2.000.000.000		•		Position des Sprites in der Layer-Ordnung
tweened	TRUE/FALSE		•		
bottom	Anzahl Pixel		•		vom oberen Bühnenrand
top	Anzahl Pixel		•		vom oberen Bühnenrand
right	Anzahl Pixel		•		vom linken Bühnenrand
left	Anzahl Pixel		•		vom linken Bühnenrand
moveableSprite	TRUE/FALSE		•		

Playback, Loop

(Properties:)

Property	Wert/ Wertebereich	Member	Sprite	nur lesen	Anmerkungen
directToStage	TRUE/FALSE	•			
trails	TRUE/FALSE		•		

Zeit, Frame-Rate*(Properties:)* keine**Quality, Blend, Alpha, Antialias, Mask***(Properties:)*

Property	Wert/ Wertebereich	Member	Sprite	nur lesen	Anmerkungen
blend	0–100		•		
blendLevel	0–255		•		
ink	0–9, 32–41		•		Standard: 0 („Kopieren“)

Farbe*(Properties:)*

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
backColor	0 – 255		•		für Sprites obsolet (siehe aber #text, #field)
foreColor	0 – 255		•		für Sprites obsolet (siehe aber #text, #field)
bgColor	Farbobjekt rgb(<i>rot</i> , <i>grün</i> , <i>blau</i>), rgb(„#RRGGBB“), paletteIndex(<i>Zahl</i>)		•		ersetzt backColor
color	wie bgColor		•		ersetzt foreColor
the paletteMapping	TRUE/FALSE				System-Property, entspricht der Einstellung „Paletten bei Bedarf anpassen“

Interaktivität, Sound, Scripts*(Properties:)*

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
scriptText	String	•			Text des Darstellerskripts oder anderer Skripttypen; zur Laufzeit nicht lesbar, aber setzbar
scriptInstanceList	z.B. [<offspring „“ 1 1100b64>]		•		Liste der Objektinstanzen; nur verfügbar, wenn der Film läuft
scriptList	z.B. [(member 2 of castlib 1), 0]		•		Liste der Darstellerreferenzen aller Behaviors eines Sprites
scriptNum	Nummer		•		Darstellernummer des ersten Behaviors

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
staticQuality	#minQuality, #maxQuality, #normalQuality		•		Darstellungsmodus
motionQuality	#minQuality, #maxQuality, #normalQuality		•		Darstellungsmodus
warpMode	#full, #partial, #none		•		Darstellungsmodus
triggerCallback	#handlername, o		•		Setzt Callback-Handler für Klicks auf Hotspots
hotSpotEnterCallback	#handlername, o		•		Setzt Callback-Handler für Mausaktion
hotSpotExitCallback	#handlername, o		•		Setzt Callback-Handler für Mausaktion
nodeEnterCallback	#handlername, o		•		Setzt Callback-Handler für VR-Statusmeldung
nodeExitCallback	#handlername, o		•		Setzt Callback-Handler für VR-Statusmeldung

- (Funktionen:)
enableHotSpot(sprite(s), HotspotID, TRUEorFALSE)
- getHotSpotRect(sprite(s), HotspotID)
- nudge(sprite(s), #Richtung) – für Richtung sind #down, #downLeft, #downRight, #left, #right, #up, #upLeft und #upRight möglich.
- ptToHotSpotID(sprite(s), Punkt)
- swing(sprite(s), pan, tilt, fieldOfView, speedToSwing)

#flash

Darstellerinhalte/-referenzen

(Darsteller:) castlibNum, filename, membernum, modified, name, number, size, thumbnail, type

(Sprite:) type

(Properties:)

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
posterFrame	Bildnummer	•			
sound	TRUE/FALSE	•	•		
url	TRUE/FALSE	•			

#flash

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
linked	TRUE/FALSE	•			linked = 0 importiert externen Darsteller
broadcastProps	TRUE/FALSE	•			wenn TRUE, wirken sich Darstelleränderungen auf sichtbare Sprites aus; wenn FALSE, sind nur neu generierte Sprites geändert

(Funktionen:)

- `member(m).showprops(), sprite(s).showprops()`

Speicher, Streaming

(für alle Darstellertypen gültig:)

(Darsteller:) `loaded, mediaReady, preload, purgePriority`

(Properties:)

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
bufferSize	Anzahl Bytes	•			
streamMode	#frame, #idle, #manual	•			
percentStreamed	0–100	•		•	
bytesStreamed	0–streamSize	•		•	
streamSize	Dateigröße in Bytes	•		•	
state	1,0,1,2,3,4	•		•	

(Funktionen:)

- `frameReady (sprite(s), BildnummerImFlashFilm)`
- `stream (member(m), numBytes)`
- `clearError()`

Größe, Ort, Sichtbarkeit, Cropping, Rotation*(für alle Darstellertypen gültig:)**(Darsteller:)* regpoint, height, rect, width*(Sprite:)* height, rect, width, stretch, loc, locH, locV, locZ, tweened, bottom, top, right, left, moveableSprite*(Properties:)*

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
imageEnabled	TRUE/FALSE	•	•		
centerRegPoint	TRUE/FALSE	•			
defaultRect	rect(o, o, breite, hoehe)	•			Standard für Darstellungsgröße, wenn neue Sprites generiert werden
defaultRectMode	#flash, #fixed	•			#fixed, um defaultRect zu nutzen
flashRect	rect(o, o, breite, hoehe)	•		•	Originalgröße des Darstellers
scale	o.o – n, Standard: 100	•	•		nicht funktional, wenn scalemode #autoSize oder #exactFit ist
originMode	#center, #topleft, #point	•	•		Fixpunkt bei Skalierung (viewScale) und Rotation; #point, um originPoint zu nutzen (Flash-Koordinaten)
originPoint	Punkt	•	•		vgl. originMode (Flash-Koordinaten)
originH	Pixelwert	•	•		(Flash-Koordinaten)
originV	Pixelwert	•	•		(Flash-Koordinaten)
viewScale	o.o – n, Standard: 100	•	•		Objekt wird größer, wenn viewScale Richtung o geht; vgl. originMode
viewPoint	Punkt	•	•		Punktangabe relativ zum originPoint; Punkt, der in der Mitte des Sprites stehen soll (Flash-Koordinaten)
viewH	Pixelwert	•	•		(Flash-Koordinaten)
viewV	Pixelwert	•	•		(Flash-Koordinaten)
scalemode	#showAll, #autoSize, #noScale, #exactFit, #noBorder	•	•		Darstellungsmodus, vgl. scale
obeyScoreRotation	TRUE/FALSE (Standard: TRUE)	•			TRUE bewirkt, daß die Sprite-Rotation im Drehbuch genutzt wird; Darstellerrotation wird ignoriert
rotation	o.o–360.o	•	•		

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
flipH	TRUE/FALSE		•		
flipV	TRUE/FALSE		•		
skew	-180.0–180.0		•		

Playback, Loop

(für alle Darstellertypen gültig;)

(Darsteller:) directToStage

(Properties:)

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
loop	TRUE/FALSE	•	•		
pausedAtStart	TRUE/FALSE	•	•		
frame	Bildnummer des Flash-Films		•		
playing	TRUE/FALSE		•	•	
playbackMode	#normal,#fixed, #lokkedStep	•	•		#fixed, um fixedRate zu nutzen
fixedRate	Bilder pro Sekunde	•	•		

(Funktionen:)

- hold *sprite(s)*
- play *sprite(s)*
- rewind *sprite(s)*
- stop *sprite(s)*
- *sprite(s).findlabel(„Labelname“)*

Zeit, Frame-Rate

(für alle Darstellertypen gültig;) keine

(Properties:)

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
frameRate	Originalbildrate des Flash-Films	•		•	Setzen der Bildrate mit playback- Mode und fixedRate
frameCount	Anzahl der Frames	•		•	

Quality, Blend, Alpha, Antialias, Mask*(für alle Darstellertypen gültig:)**(Sprite:) blend, blendLevel, ink**(Properties:)*

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
quality	#high, #low, #autoHigh, #autoLow	•	•		Darstellungsmodus
static	TRUE/FALSE	•	•		bei Ein-Bild-Flash-Filmen oder angehaltenen Animationen spart static = 1 Performance

Farbe*(für alle Darstellertypen gültig:)**(Sprite:) backColor, foreColor, bgColor, color**(Properties für #flash:) keine***Interaktivität, Sound, Scripts***(Darsteller:) scriptText**(Sprite:) scriptInstanceList, scriptNum**(Properties:)*

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
buttonsEnabled	TRUE/FALSE (Standard: TRUE)	•	•		visuelle Komponente von Buttons (Bildänderung) EIN/AUS
actionsEnabled	TRUE/FALSE (Standard: TRUE)	•	•		Button- und Frame-Aktionen EIN/AUS
eventPassMode	#passAlways, #passNever, #passButton, #passNotButton	•	•		regelt, ob und wann Mausektionen vom Flash-Film wieder an Director weitergeleitet werden
clickMode	#opaque, #boundingbox, #object	•	•		regelt, welche Bereiche des Flash-Films empfänglich sind für Mausevents; für #opaque muß der Ink-Modus des Sprites auf „Hintergrund transparent“ stehen
mouseOverButton	TRUE/FALSE	•	•	•	
soundMixMedia	TRUE/FALSE (Standard: TRUE)				Systemproperty; TRUE für Sound-mixing durch Director

(Funktionen:)

- `hitTest(sprite(s), BühnenPunkt)` – #background, #normal, #button
(D7.02/Flash 4: zusätzlich #editText)
- zusätzlich in D7.02/Flash 4:
- `getVariable(sprite(s), variablenName)`
- `setVariable(sprite(s), variablenName, neuerWert)`

Systemfunktionen

(Funktionen:)

- `flashToStage(sprite(s), FlashPunkt)`
- `stageToFlash(sprite(s), BühnenPunkt)`
- `hitTest(sprite(s), BühnenPunkt)` – #background, #normal, #button
(D7.02/Flash 4: zusätzlich #editText)

#animGIF

Darstellerinhalte/-referenzen

(für alle Darstellertypen gültig:)

(Darsteller:) `castlibNum`, `filename`, `membernum`, `modified`, `name`, `number`, `size`, `thumbnail`, `type`

(Sprite:) `type`

(Properties:)

Aktuelle Ergänzung:
Flash 4-Unterstützung
(ab Director 7.02)

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
URL	URL	•			
linked	TRUE/FALSE	•			
picture	<Objekt>	•			
media	<Objekt>	•			

Speicher, Streaming

(für alle Darstellertypen gültig:)

(Darsteller:) `loaded`, `mediaReady`, `preload`, `purgePriority`

(Properties für #animGIF:) keine

Zeit, Frame-Rate

(für alle Darstellertypen gültig;) keine

(Properties für #animGIF:) siehe Playback

Quality, Blend, Alpha, Antialias, Mask

(für alle Darstellertypen gültig;)

(Sprite:) blend, blendLevel, ink

(Properties für #animGIF:) keine

Farbe

(für alle Darstellertypen gültig;)

(Sprite:) backColor, foreColor, bgColor, color

(Properties für #animGIF:) keine

Interaktivität, Sound, Scripts

(für alle Darstellertypen gültig;)

(Darsteller:) scriptText

(Sprite:) scriptInstanceList, scriptNum

(Properties für #animGIF:) keine

Systemfunktionen

(Funktionen:) keine

#vectorshape

Darstellerinhalte/-referenzen

(für alle Darstellertypen gültig;)

(Darsteller:) castlibNum, filename, membernum, modified, name, number, size, thumbnail, type

(Sprite:) type

(Properties:)

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
closed	TRUE/FALSE	•			
strokeWidth	Breite in Pixeln als #float	•			

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
fillMode	#gradient, #none, #solid	•			
gradientType	#radial, #linear	•			
fillScale	0–100	•			
fillDirection	0.0–360.0	•			
fillOffset	Punkt	•			
fillCycles	1–7	•			Anzahl der Verlaufsdurchläufe
broadcastProps	TRUE/FALSE	•			wenn TRUE, wirken sich Darsteller-Änderungen auf sichtbare Sprites aus; wenn FALSE, sind nur neu generierte Sprites geändert
		<p>(Funktionen:)</p> <p><i>member(m).showprops(), sprite(s).showprops()</i></p> <p>Speicher, Streaming</p> <p>(für alle Darstellertypen gültig:)</p> <p>(Darsteller:) loaded, mediaReady, preload, purgePriority</p> <p>(Properties für #vectorshape:) keine</p> <p>Größe, Ort, Sichtbarkeit, Cropping, Rotation</p> <p>(für alle Darstellertypen gültig:)</p> <p>(Darsteller:) regpoint, height, rect, width</p> <p>(Sprite:) height, rect, width, stretch, loc, locH, locV, locZ, tweened, bottom, top, right, left, moveableSprite</p> <p>(Properties:)</p>			
Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
imageEnabled	TRUE/FALSE	•	•		
centerRegPoint	TRUE/FALSE	•			
defaultRect	rect(o, o, breite, hoehe)	•			Standard für Darstellungsgröße, wenn neue Sprites generiert werden
defaultRectMode	#flash, #fixed	•			#fixed, um defaultRect zu nutzen
flashRect	rect(o, o, breite, hoehe)	•		•	Originalgröße des Darstellers
scale	0.0 – n	•	•		nicht funktional, wenn scalemode #autoSize oder #exactFit ist

#vectorshape

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
originMode	#center, #topleft, #point	•	•		Fixpunkt bei Skalierung (viewScale) und Rotation; #point, um originPoint zu nutzen (Flash-Koordinaten)
originPoint	Punkt	•	•		vgl. originMode (Flash-Koordinaten)
originH	Pixelwert	•	•		(Flash-Koordinaten)
originV	Pixelwert	•	•		(Flash-Koordinaten)
viewScale	0.0 – n, Standard: 100	•	•		Objekt wird größer, wenn viewScale Richtung 0 geht; vgl. originMode
viewPoint	Punkt	•	•		Punktangabe relativ zum originPoint; Punkt, der in der Mitte des Sprites stehen soll (Flash-Koordinaten)
viewH	Pixelwert	•	•		(Flash-Koordinaten)
viewV	Pixelwert	•	•		(Flash-Koordinaten)
scaleMode	#showAll, #autoSize, #noScale, #exactFit, #noBorder	•	•		Darstellungsmodus, vgl. scale #vw# (Abbildung)
rotation	0.0–360.0		•		
flipH	TRUE/FALSE		•		
flipV	TRUE/FALSE		•		
skew	-180.0–180.0		•		

Playback, Loop

(für alle Darstellertypen gültig:)

(Darsteller:) directToStage

(Properties für #vectorshape:) keine

Zeit, Frame-Rate

(für alle Darstellertypen gültig:) keine

(Properties für #vectorshape:) keine

Quality, Blend, Alpha, Antialias, Mask

(für alle Darstellertypen gültig:)

(Sprite:) blend, blendLevel, ink

(Properties:)

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
antiAlias	TRUE/FALSE	•	•		
static	TRUE/FALSE	•	•		

Farbe*(für alle Darstellertypen gültig:)**(Sprite:)* backColor, foreColor, bgColor, color*(Properties:)*

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
strokeColor	rgb(<i>r,g,b</i>), rgb(„#rrggbb“), paletteIndex(<i>i</i>)	•			Linienfarbe
fillColor	wie strokeColor	•			Füllungsfarbe
endColor	wie strokeColor	•			Endfarbe (bei Verlauf)
backgroundColor	wie strokeColor	•			Farbe des Hintergrunds

Interaktivität, Sound, Scripts*(für alle Darstellertypen gültig:)**(Darsteller:)* scriptText*(Sprite:)* scriptInstanceList, scriptNum*(Properties für #vectorshape:)* keine*(Funktionen:)*

hitTest(sprite(s), BühnenPunkt) – #background, #normal

Systemfunktionen*(Funktionen:)*

- flashToStage(sprite(s), Punkt)
- stageToFlash(sprite(s), Punkt)
- hitTest(sprite(s), BühnenPunkt) – #background, #normal

Vertexmanipulationen

(Properties:)

Property	Wert/Wertebereich	Member	Sprite	nur lesen	Anmerkungen
vertexList	[[#vertex: <i>Punkt</i> {, #handle1: <i>Punkt</i> {, #handle2: <i>Punkt</i> {}}, ...] -- <i>Punkte</i> als #float	•			Es müssen immer ganze Vertexlisten übergeben werden; vgl. vertex
vertex[n]	<Objekt>	•			Chunk-Expression; ermöglicht das Adressieren von einzelnen Vertexpunkten und ihrer Handles

(Funktionen:)

- `addVertex(member(m), numPosition, Punkt {,controlLocH, controlLocV})`
- `deleteVertex(member(m), numPosition)`
- `moveVertex(member(m), numPosition, xChange, yChange)`
- `moveVertexHandle(member(m), numPosition, numHandle, xChange, yChange)`

#bitmap

Darstellerinhalte/-referenzen

(für alle Darstellertypen gültig:)

(Darsteller:) castlibNum, filename, membernum, modified, name, number, size, thumbnail, type

(Sprite:) type

(Properties:)

Property	Wert	Member	Sprite	nur lesen	Anmerkungen
picture	<Objekt>	•			
media	<Objekt>	•			

Speicher, Streaming

(für alle Darstellertypen gültig:)

(Darsteller:) loaded, mediaReady, preload, purgePriority

(Properties für #bitmap:) keine