5 Programming with Animations

5.1 Animated Graphics: Principles and History

5.2 Types of Animation

5.3 Programming Animations

5.4 Design of Animations

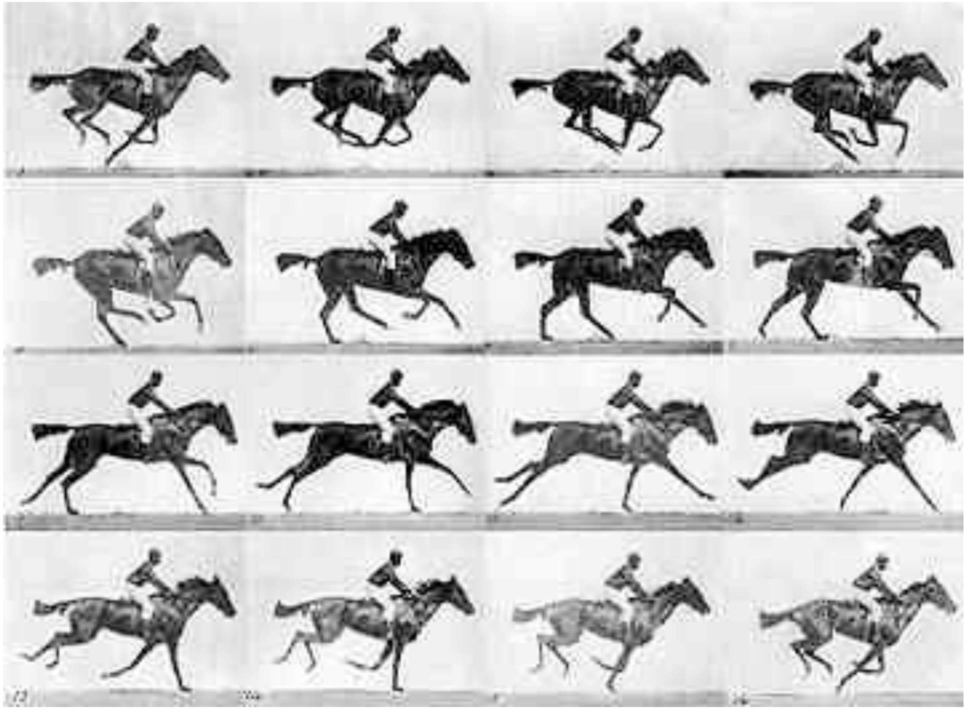
5.5 Game Physics

Literature:

P. Ackermann: Developing Object-Oriented Multimedia Software based on the MET++ Application Framework, dpunkt 1996

Eadweard Muybridge: Chronofotografie

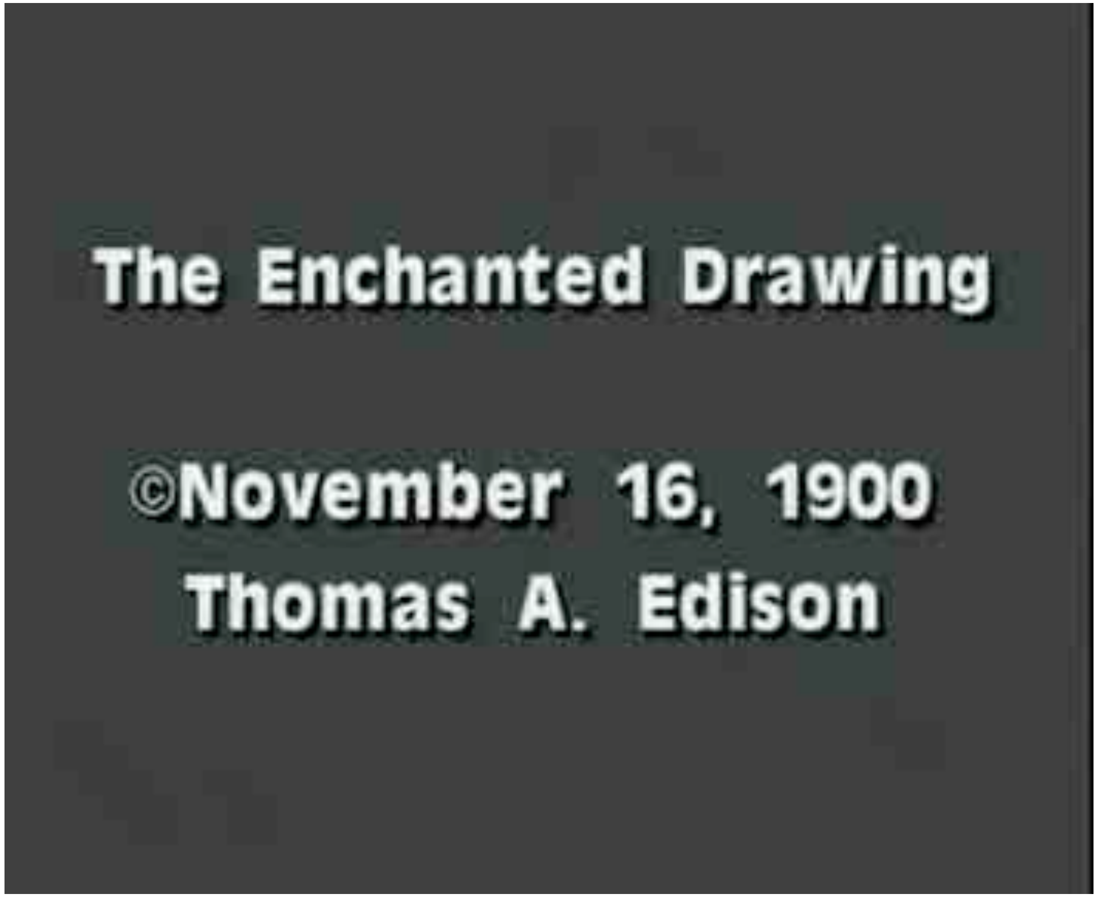
- 1830 – 1904



Quelle: Wikipedia

J. Stuart Blackton: The Father of Animation

- 1875 – 1941
- Became “rapid drawing cartoonist” for Thomas A. Edison



The Enchanted Drawing
1900

Problem: How to Create SO Many Pictures?

Drawing work for “Gertie the Dinosaur”

Winsor McKay: Character Animation

- 1867 – 1934

Gertie the Dinosaur
1914

First character animation
First keyframe animation

“He devised what he called the "McCay Split System", ...
Rather than draw each frame in sequence, he would start by drawing Gertie's key poses, and then go back and fill in the frames between.”
(Wikipedia)

Walt Disney: Animation Industry

- 1901 – 1966

Pencil

Pen

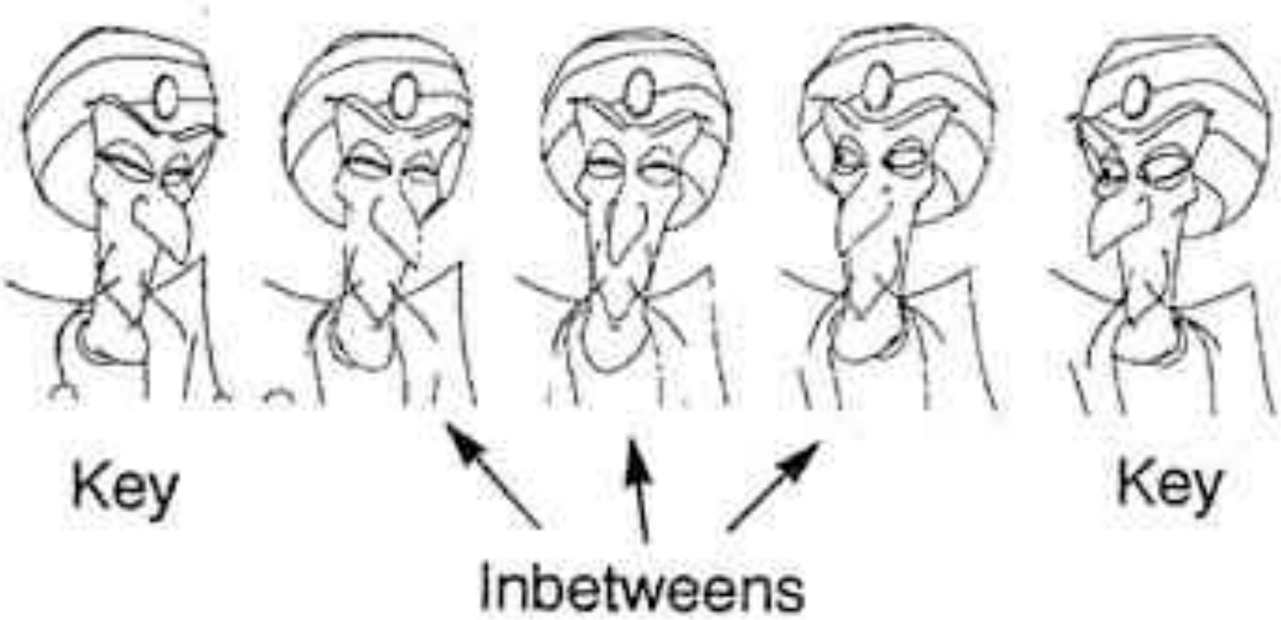
Ink

Source: Midori Kitagawa, http://atec.utdallas.edu/midori

In-Between Drawing

- *Key frames:* Define the start and end points of a smooth transition
- *In-between frames:* Filled in to create the transition

Traditional hand-drawn animation:
Work split between senior artist and assistant



Source: Midori Kitagawa, http://atec.utdallas.edu/midori

Animation by Interpolation

- Key frame:
  - Contains manually defined objects & object attributes
- In-between frame:
  - Object attributes computed automatically
- Computation of attribute values:
  - Discrete interpolation:
    - » Start and end value given
    - » Intermediate position given by frame number
  - E.g. (linear interpolation):

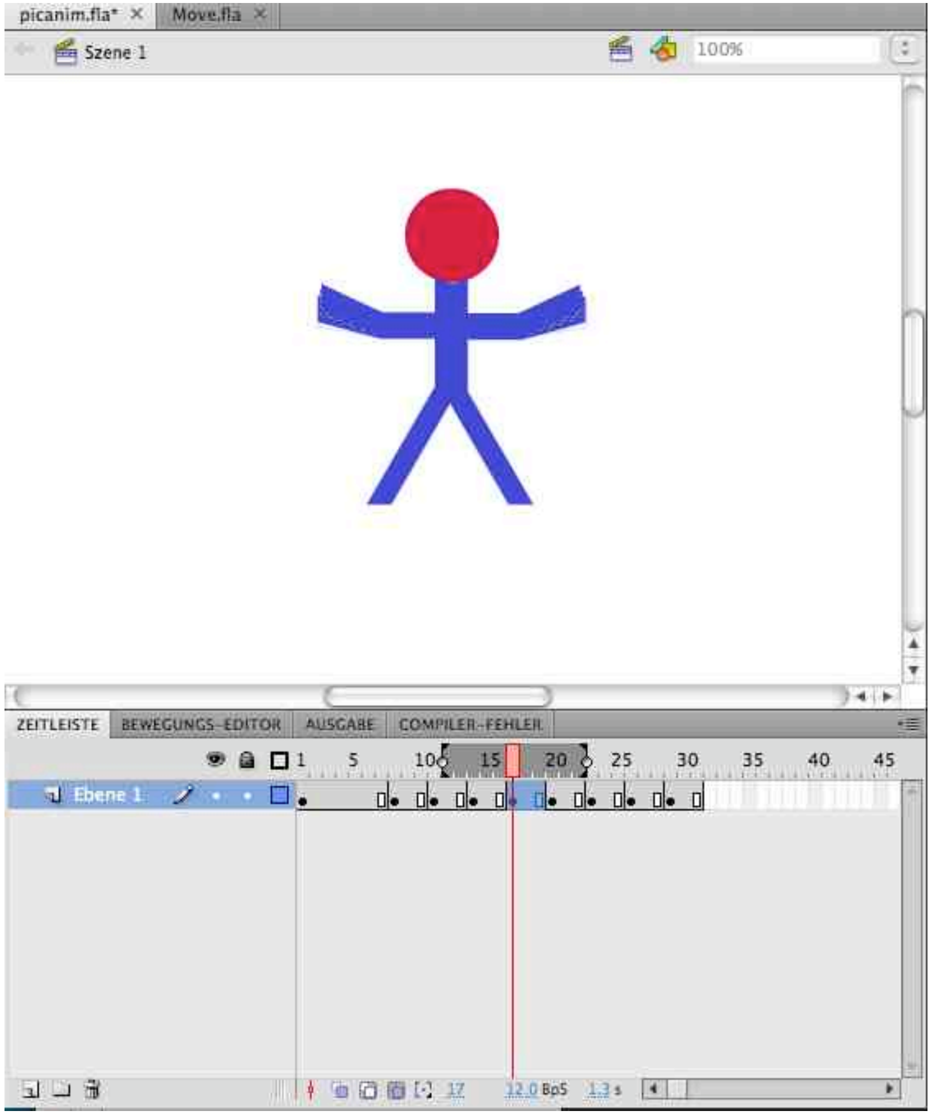
    delta = (*end* – *start*) / *steps*

    value(*i*) = *start* + *delta* * *i*

5 Programming with Animations

5.1 Animated Graphics: Principles and History
5.2 Types of Animation
5.3 Programming Animations
5.4 Design of Animations
5.5 Game Physics

Frame-By-Frame Animation



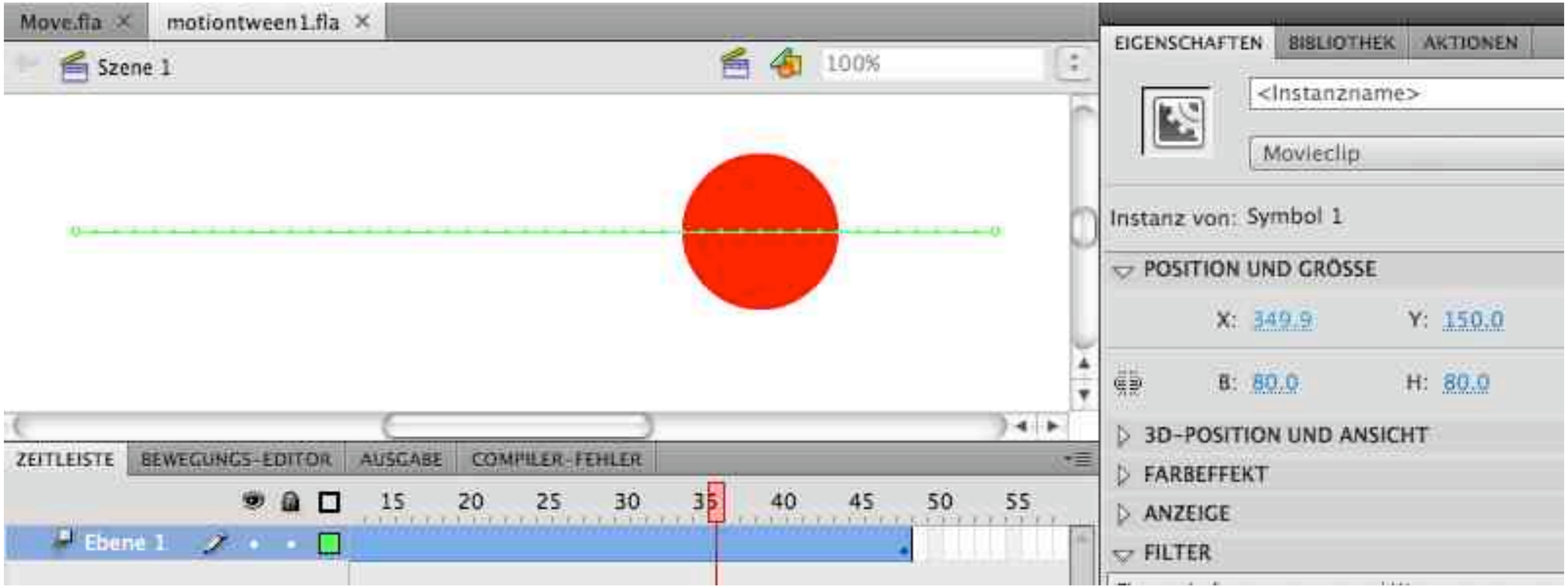
Each image is drawn manually

Special tools may be used for previewing the effect
(onion skinning)

Keyframe Animation: Motion Tween in Flash

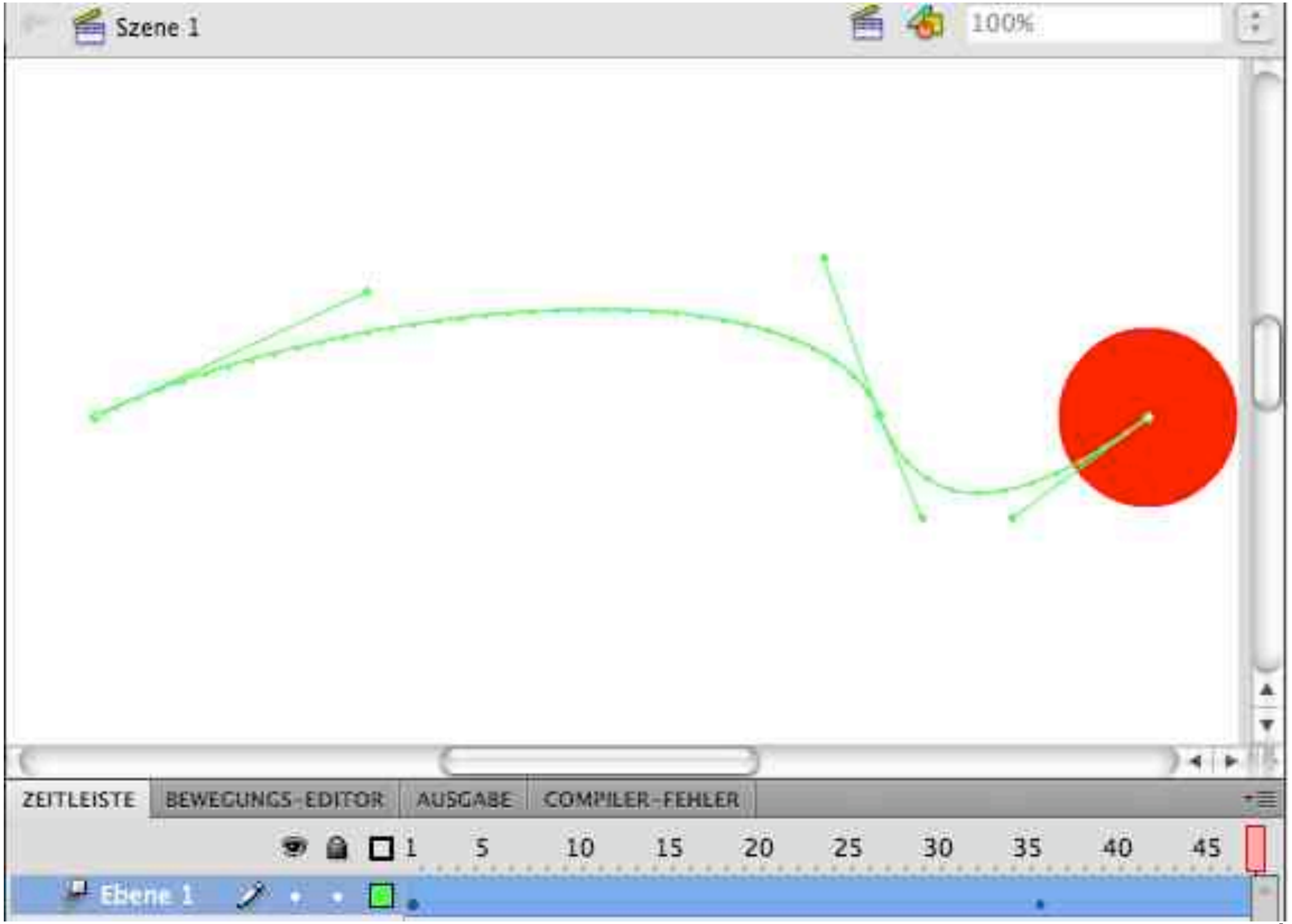
- Properties of a (2D) object manipulated by motion tween:
  - Position (x and y)
  - Rotation (z)
  - Skew/Shear *(Neigung)*
  - Size
  - Colour effects
- Basic idea of graphically creating a motion tween:
  - Place an object (instance!) on a separate layer
  - Invoke “Create Motion Tween” (context menu)
  - Readjust property values graphically or by inspector dialogue for end frame
- Property key frames:
  - Intermediate frames with individually defined object properties
- Motion path:
  - Bezier curve, can be adjusted graphically

Example: Motion Tween in Flash (1)



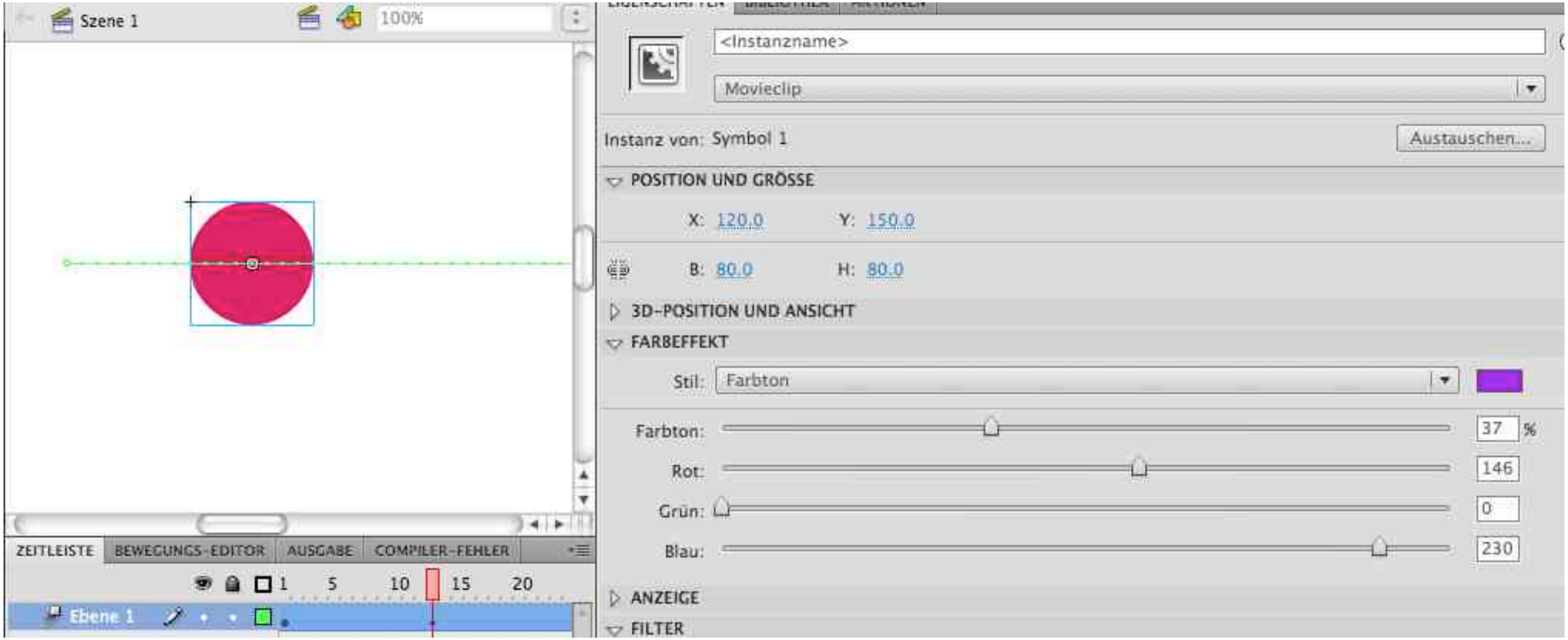
motiontween0.fla

Example: Motion Tween in Flash (2)

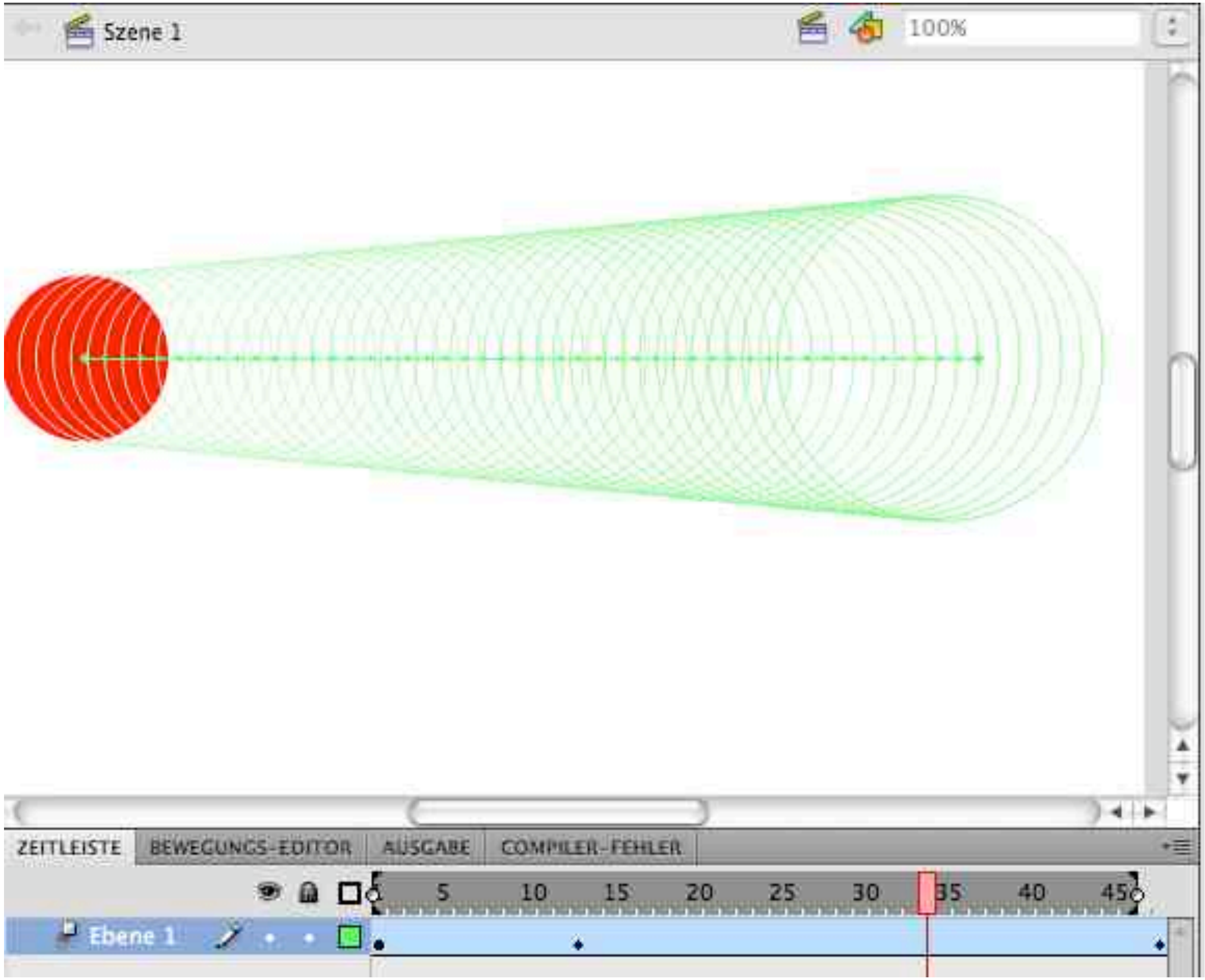


motiontween1.fla

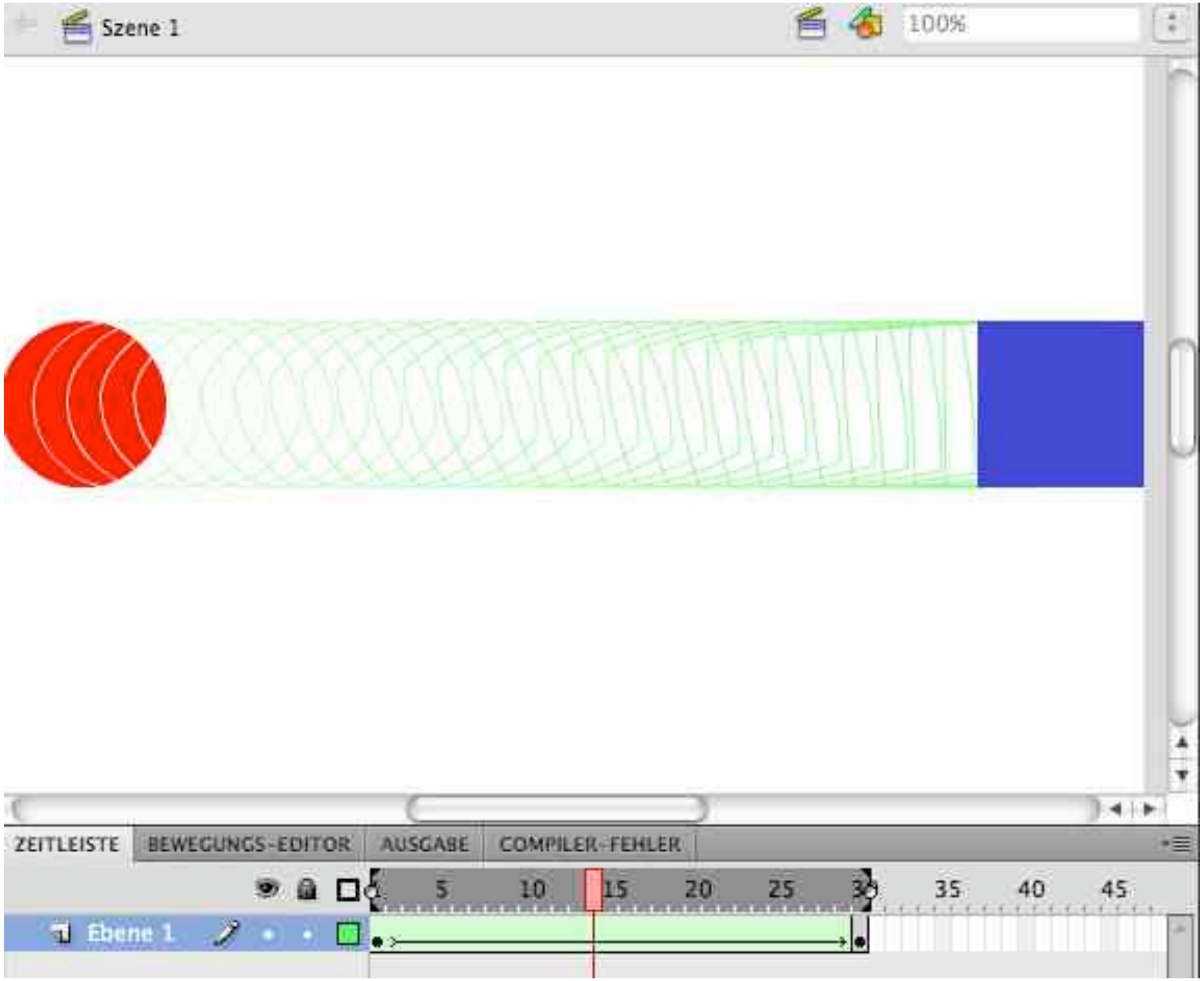
Example: Tweening Colours in Flash



Example: Tweening Object Size in Flash



Example: Shape Tweening *(Morphing)* in Flash



Shape tweening interpolates between geometric shapes

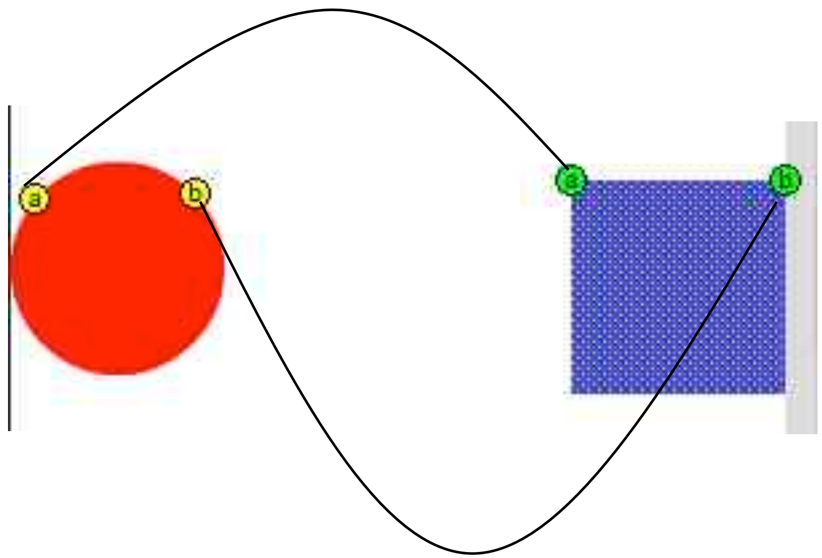
Different way of creation:

One layer containing two key frames with the two shapes

Example: Shape Hints (Flash)

Shape hints (Formmarker) enable fine control of shape tweening

- Pair of (start/end) points to be mapped on each other in transformation



5 Programming with Animations

5.1 Animated Graphics: Principles and History

5.2 Types of Animation

5.3 Programming Animations

5.4 Design of Animations

5.5 Game Physics

Literature:

P. Ackermann: Developing Object-Oriented Multimedia Software based on the MET++ Application Framework, dpunkt 1996

Linear Interpolation of Position (Python/Pygame)

```
xstart = 40
xend = 600

steps = 80 #Number of steps
deltax = (xend - xstart)/steps
frame_no = 1
x = xstart
y = 240

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    pygame.draw.rect(screen,white,Rect((0,0),(scr_width,scr_height)))
    pygame.draw.circle(screen,red,(x,y),40)

    if frame_no < steps+1:
        x = xstart + deltax*frame_no
        frame_no += 1

    pygame.display.update()
```

Speed of animation depends on computing speed

Absolute positioning of objects gives precise control

Interpolation using Fixed Frame Rate

```
xstart = 40
xend = 600

framerate = 30 #frames per second
steps = 80 #Number of steps
deltax = (xend - xstart)/steps

clock = pygame.time.Clock()
x = xstart
y = 240

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    pygame.draw.rect(screen,white,Rect((0,0),(scr_width,scr_height)))
    pygame.draw.circle(screen,red,(x,y),40)

    timepassed = clock.tick(framerate)

    if x+40 < screen_width:
        x += deltax

    pygame.display.update()
```

Speed of animation relative to frame rate

Relative positioning of objects leads to simple code

Computation of Speed

- Frame rate f, e.g. $f = 30$ frames/s
  - Time between frames $t_f = 1/f$, e.g. $t_f = 1/30$ s = 0.033 s
- Number of in-between steps s, e.g. $s = 80$
- Distance d, e.g. $d = 560$ px
- Distance of motion per frame: $d_f = d/s$, e.g. $d_f = 560/80$ px = 7 px
- Speed of animation motion v:

  $v = d_f / t_f$

  E.g. $v = 7 / (1/30) = 7 \cdot 30$ px/s = 210 px/s

- Alternative way of specifying motion timing:
  Motion speed is defined, distance per frame is computed

  $d_f = t_f \cdot v$

  $s = d / d_f = d / (t_f \cdot v) = (f \cdot d)/v$

Interpolation with Fixed Frame Rate and Speed

```
xstart = 40
xend = 600

framerate = 30 #frames per second
speed = 210 #pixels per second
clock = pygame.time.Clock()

x = xstart
y = 240

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    pygame.draw.rect(screen,white,Rect((0,0),(scr_width,scr_height)))
    pygame.draw.circle(screen,red,(x,y),40)

    timepassed_secs = clock.tick(framerate)/1000.0

    if x+40 < screen_width:
        x += timepassed_secs*speed

    pygame.display.update()
```

Interpolating Colors

```
red = (255,0,0)
blue = (0,0,255)
white = (255,255,255)

def blend_color (color1,color2,blend_factor):
    red1, green1, blue1 = color1
    red2, green2, blue2 = color2
    red0 = red1+(red2-red1)*blend_factor
    green0 = green1+(green2-green1)*blend_factor
    blue0 = blue1+(blue2-blue1)*blend_factor
    return int(red0), int(green0), int(blue0)


blend_color(red,blue,colorfactor)
```

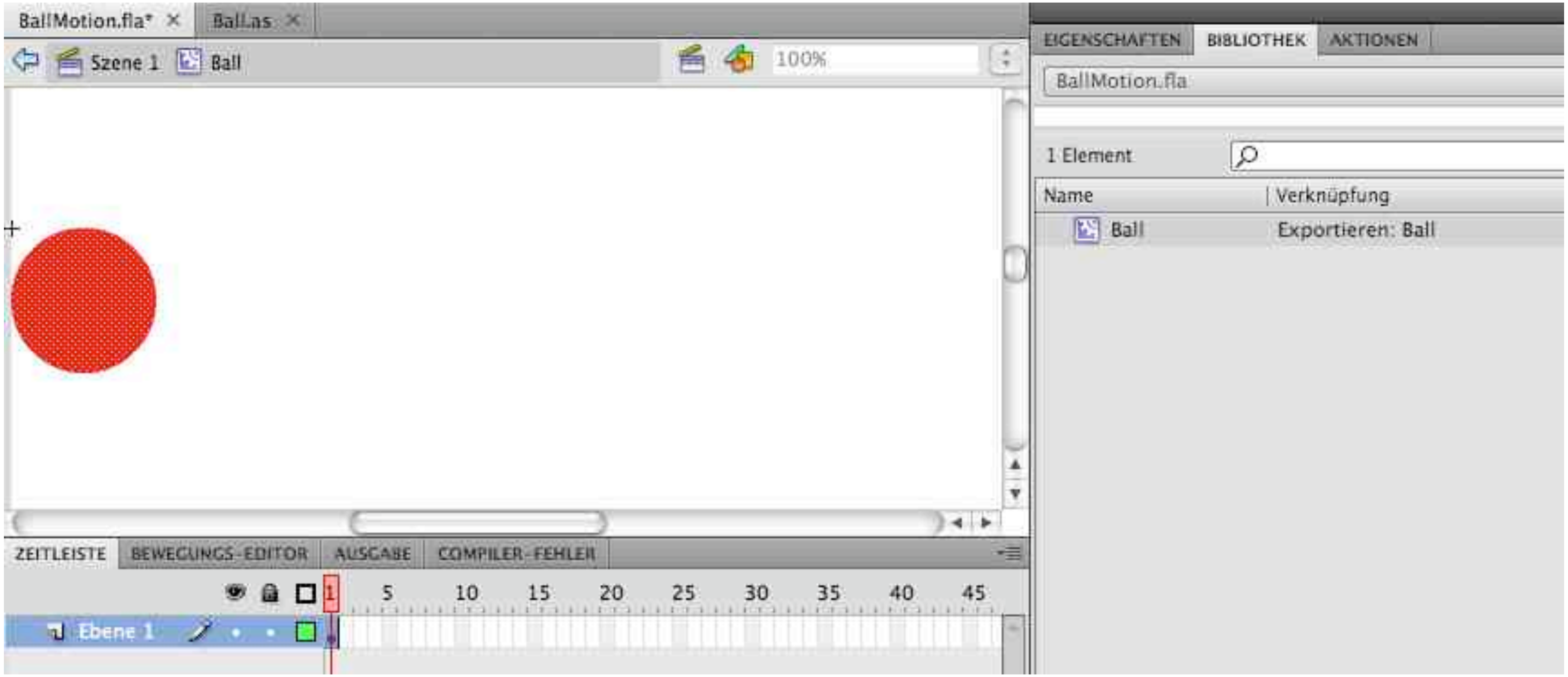
Interpolating Colors and Size

```
...
x = xstart
y = 240
steps = framerate*(xend-xstart)/speed
sizefactor = 1
colorfactor = 0

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    pygame.draw.rect(screen,white,Rect((0,0),(scr_width,scr_height)))
    pygame.draw.circle(screen,blend_color(red,blue,colorfactor),
         (x,y),40*sizefactor)

    timepassed_secs = clock.tick(framerate)/1000.0

    if x+80 < screen_width:
        x += timepassed_secs*speed
        sizefactor += 1.0/steps
        colorfactor += 1.0/steps

    pygame.display.update()
```

Frame-Dependent Animation in Flash

- Animation:
  - Modification of object attributes dependent on time / current frame
  - How to flexibly react on progress of time?
- *ENTER_FRAME* event:
  - Fired every time a new frame is displayed
  - Requires a special event handler to be registered
- Object-oriented program logic:
  - All objects have their local methods for dealing with changes
    - E.g. by moving their position
    - **`MovieClip`** subclasses inherit e.g. **`x`** and **`y`** properties
  - Enter frame event handler needs to call all necessary update methods

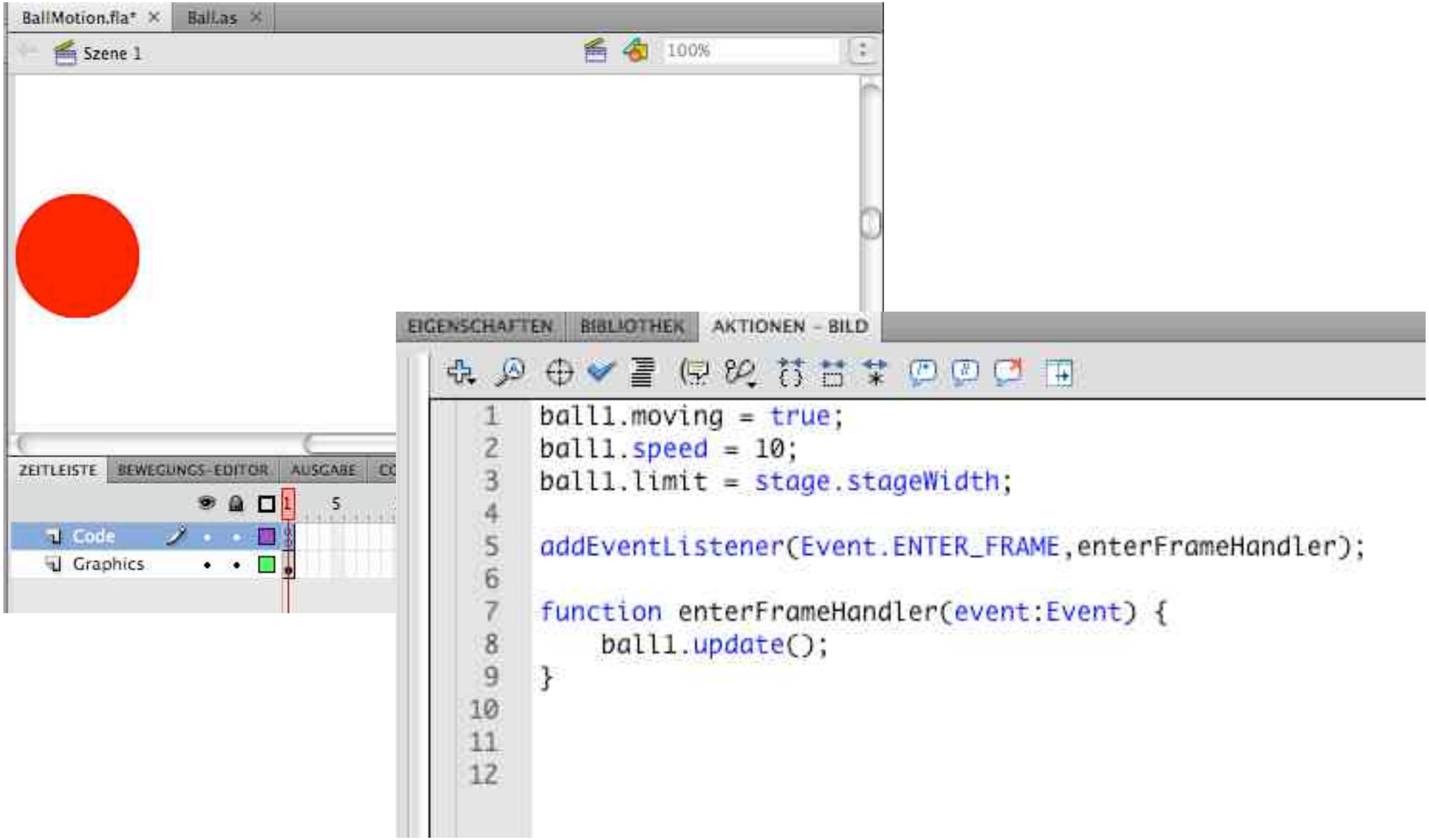
Example: Frame-Dependent Animation in Flash (1)



Example: Frame-Dependent Animation in Flash (2)

```
package {

    import flash.display.*;

    public class Ball extends MovieClip {

        public var speed:Number=0;
        public var moving:Boolean=false;
        public var limit:Number=0;

        public function update() {
            if (moving) {
                x+=speed;
                if ((x <= 0) || (x+width >= limit))
                    speed = -speed;
            }
        }
    }

}
```

Example: Frame-Dependent Animation in Flash (3)

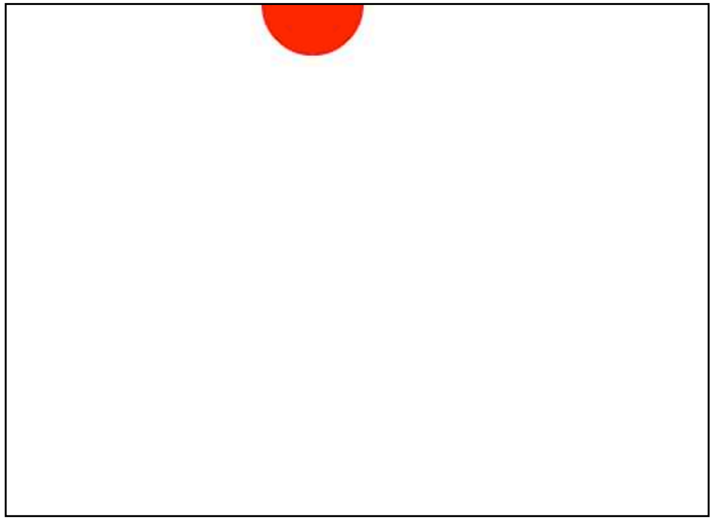


Adding Vertical Movement

```
package { import flash.display.*;
   public class Ball extends MovieClip {
      public var speed:Number=0;
      public var moving:Boolean=false;
      public var limit:Number=0;
      public var jump:Number = 0;
      public var toRight = true;
      public var inLeftHalf:Boolean = true;
      function update() {
         if (moving) {
            x+=speed;
            if ((x <= 0) || (x+width >= limit)) {
               speed = -speed;
               toRight = !toRight;
         }
         inLeftHalf = (x+width)*2 <= limit;
         if ((inLeftHalf && toRight) || (!inLeftHalf && !toRight))
            y -= jump;
         else
            y += jump;
      }
   }
}
```

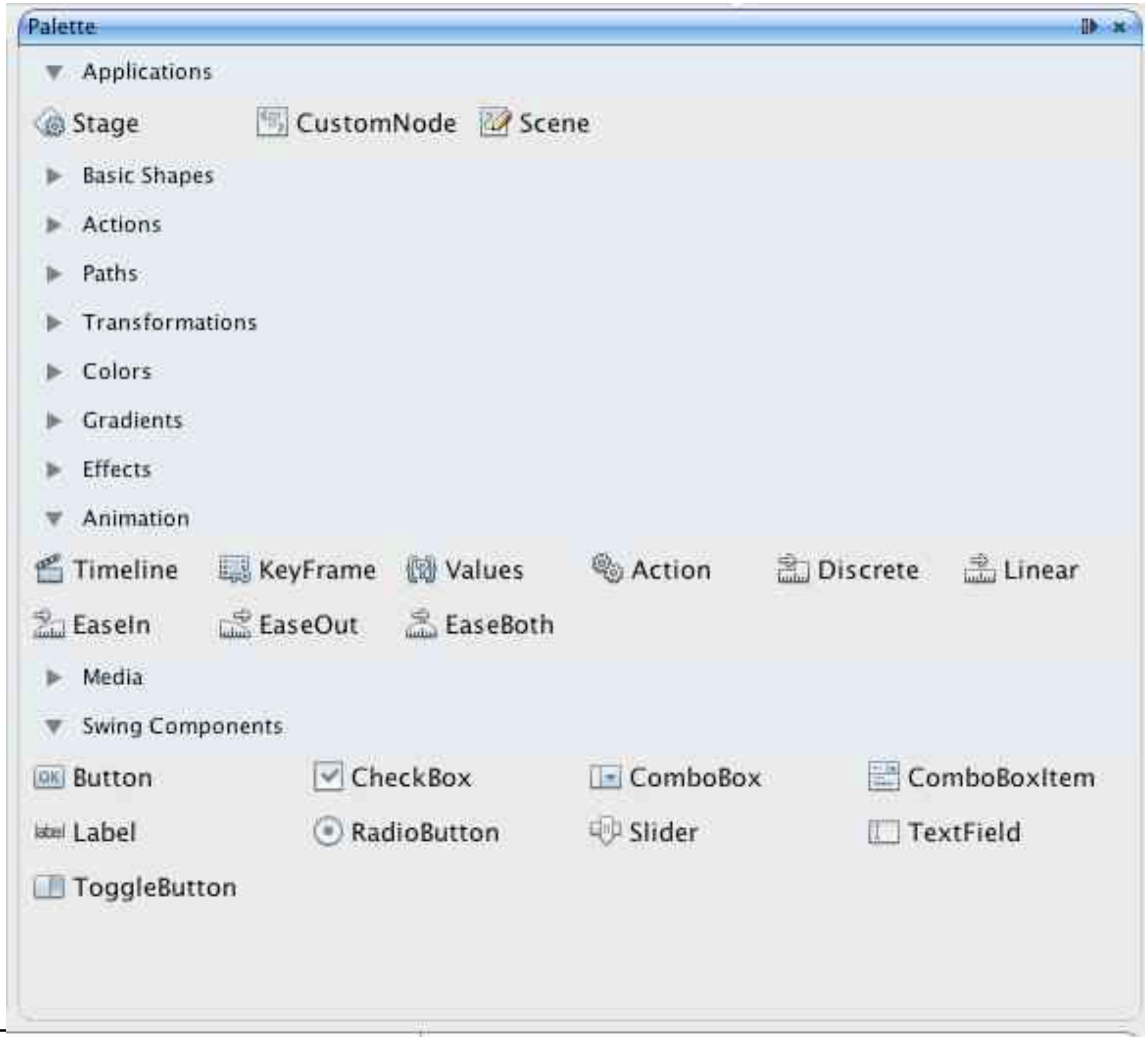
Collision Detection

- Moving objects may meet other objects and boundaries
  - *Collision detection* algorithm is responsible for detecting such situations
- Simple collision detection:
  - Width and/or height, calculated from expected position, is beyond some limit
- Potential problem:
  - Rounding errors may conceal collision event!



Animation in JavaFX

- JavaFX contains pre-defined animation templates
- Key idea is the mapping from timeline values to actual object values



Non-Linear Interpolation

- **`EaseIn / EaseOut / EaseBoth:`**
  - Methods of “slowing down” and speeding uo
  - Frequently used (in small proportions) in sorts
  - Idea: Start slowly, speed up, “cruise”, slow down, end smoothly