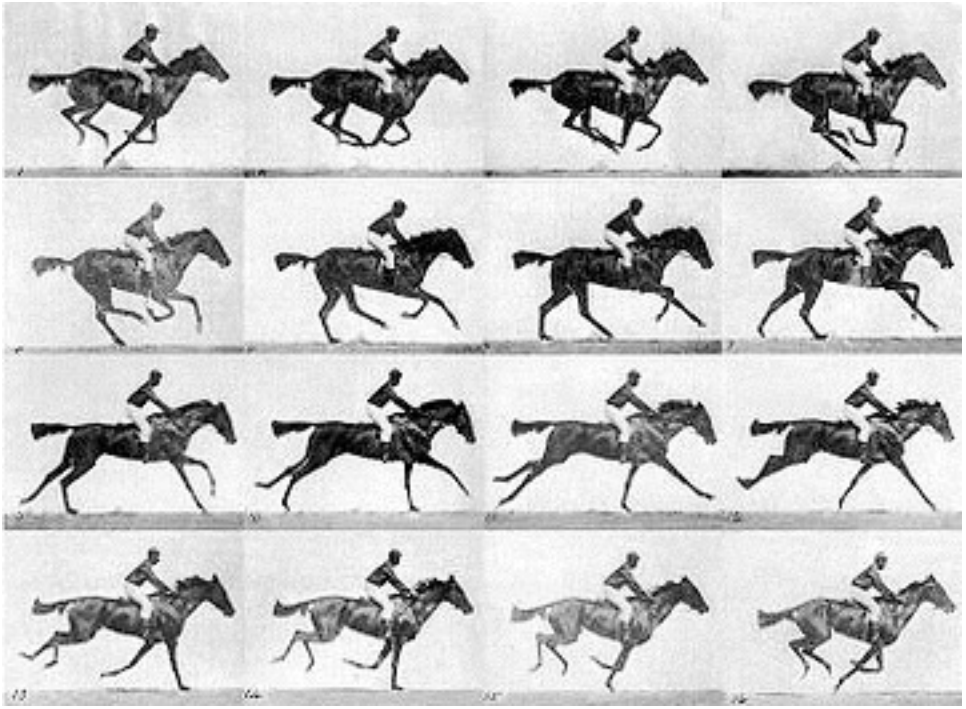
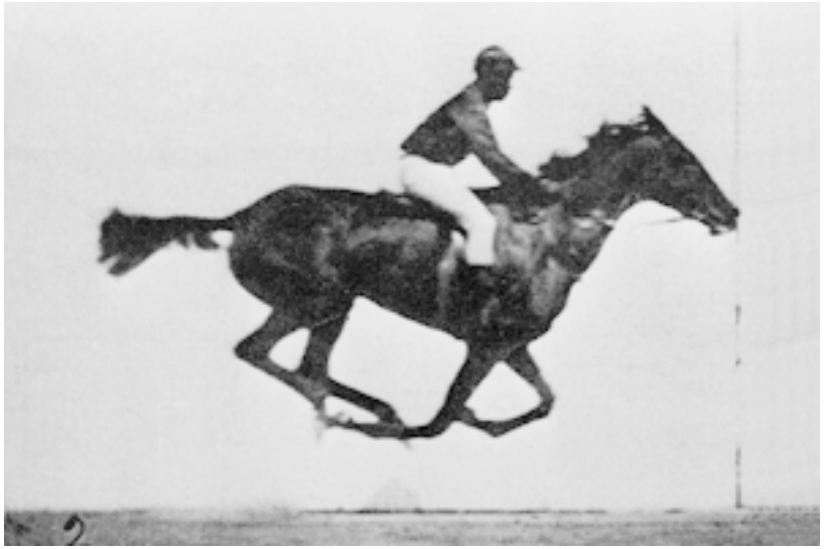
7 Programming with Animations

Eadweard Muybridge: Chronofotografie

- 1830 – 1904

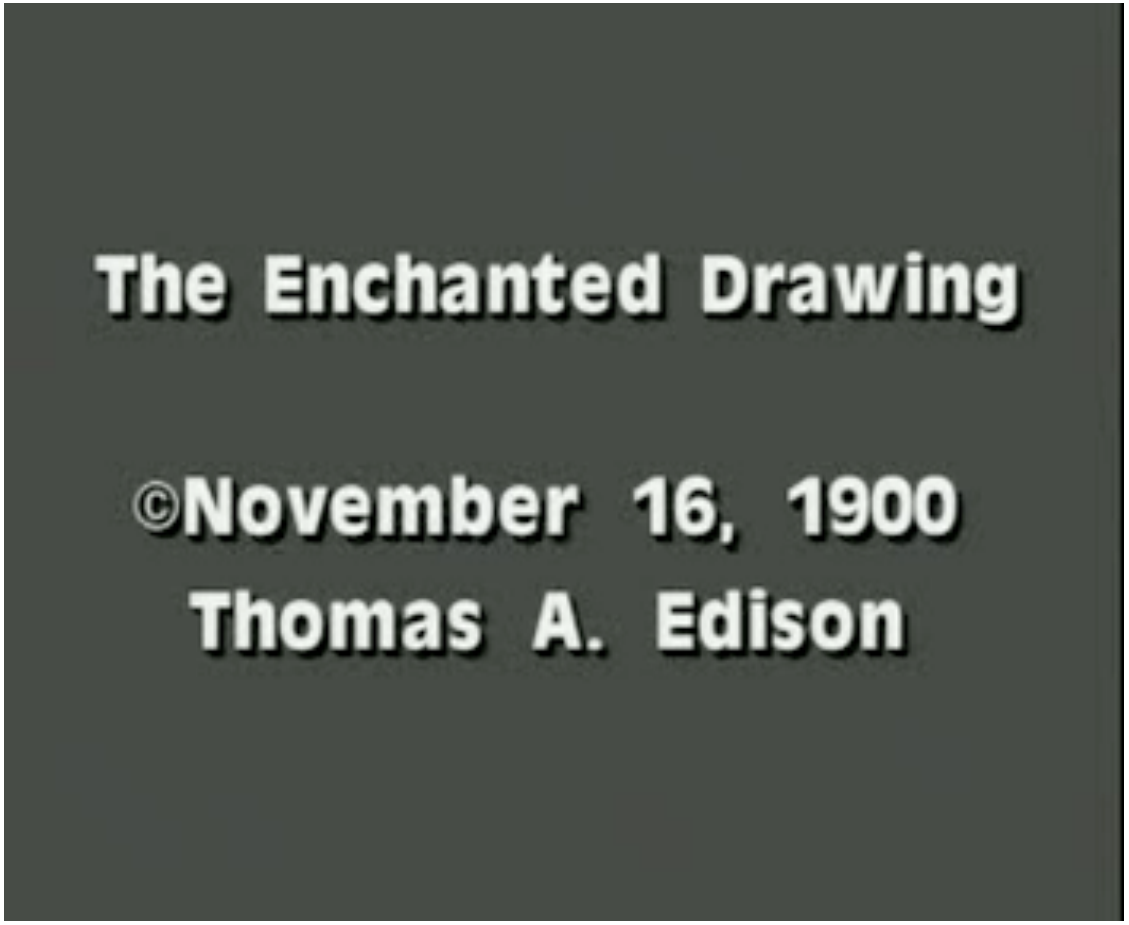




Quelle: Wikipedia

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

Winsor McKay:
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

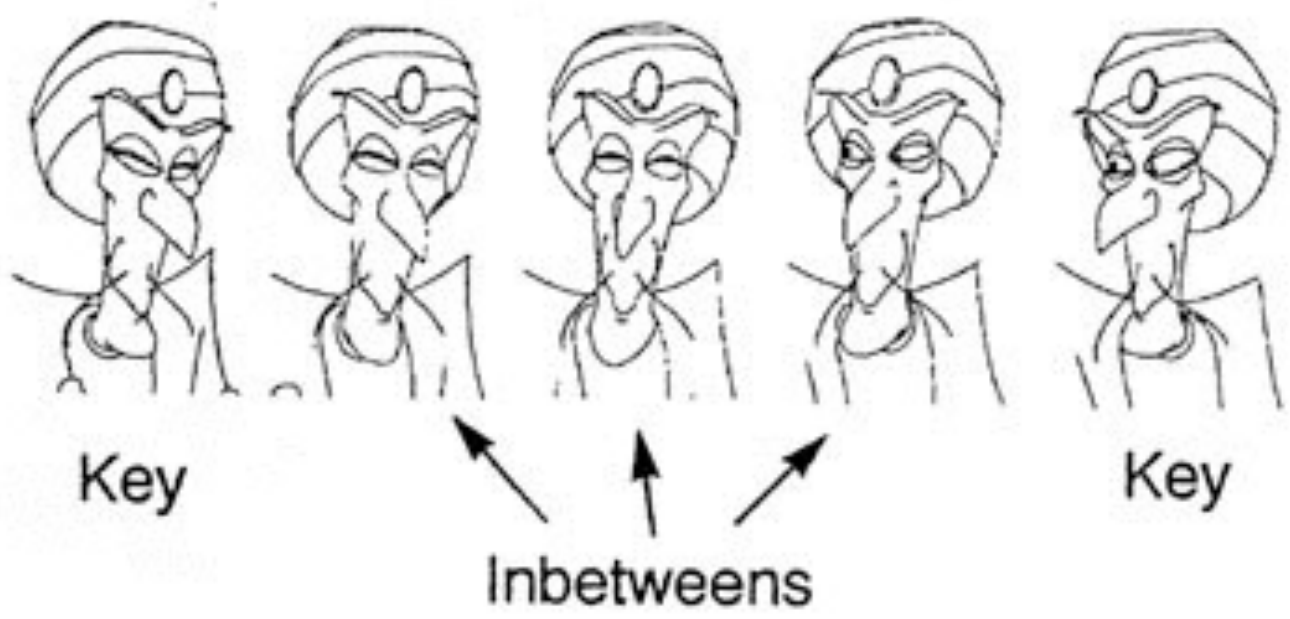
Ink

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

7 Programming with Animations

7.1 Animated Graphics: Principles and History

7.2 Types of Animation 

7.3 Programming Animations: Interpolation

7.4 Design of Animations

Literature:

http://gamedevelopment.tutsplus.com/tutorials/
an-introduction-to-spritesheet-animation--gamedev-13099

Frame-By-Frame Animation

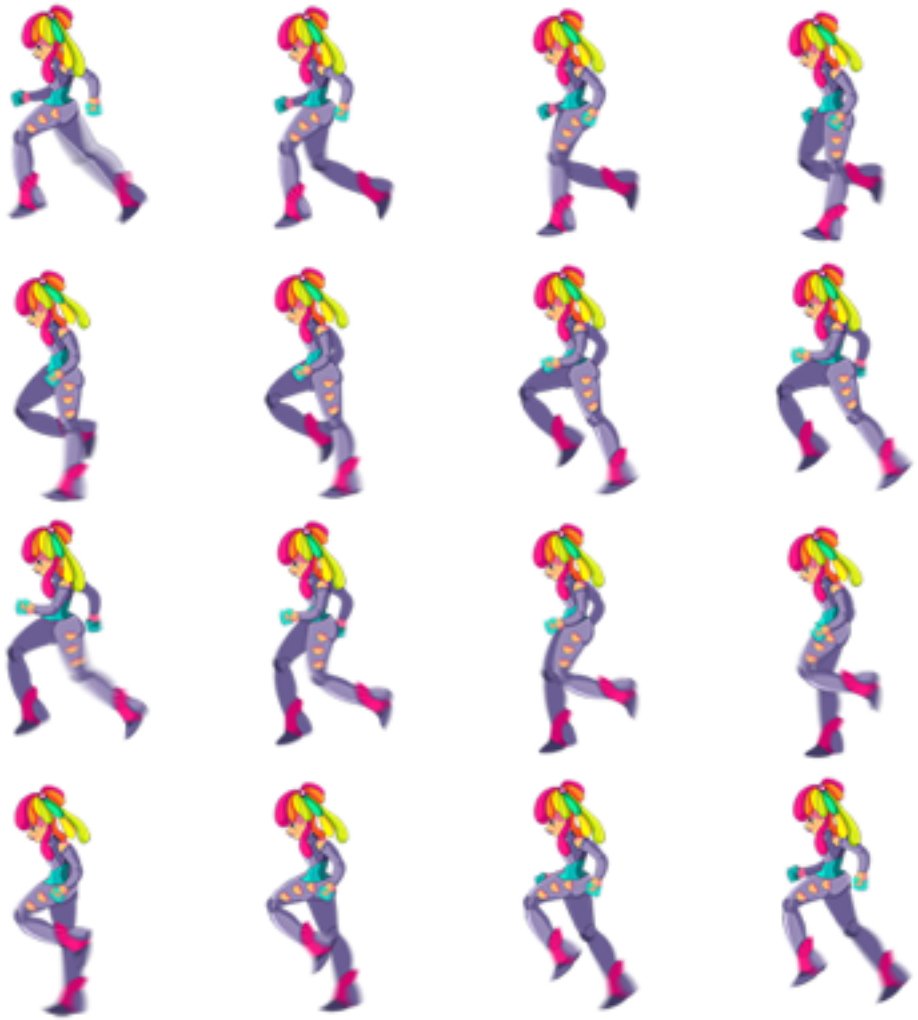
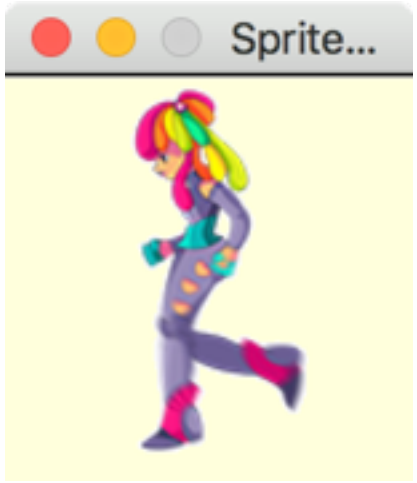


Image: tutsplus.com

- Creating a custom picture for each phase of animation
- Pictures are cycled for display
- Usually, end of sequence smoothly fits with start
- Bitmap pictures mostly
- Complete sequence on one picture file: *Sprite Sheet*
- Very frequently used
- Specific tools exist to create sprite sheets
  - E.g. TexturePacker
  - Specific classes in Cocos2d-x

Fram-By-Frame Animation with Pygame Sprites



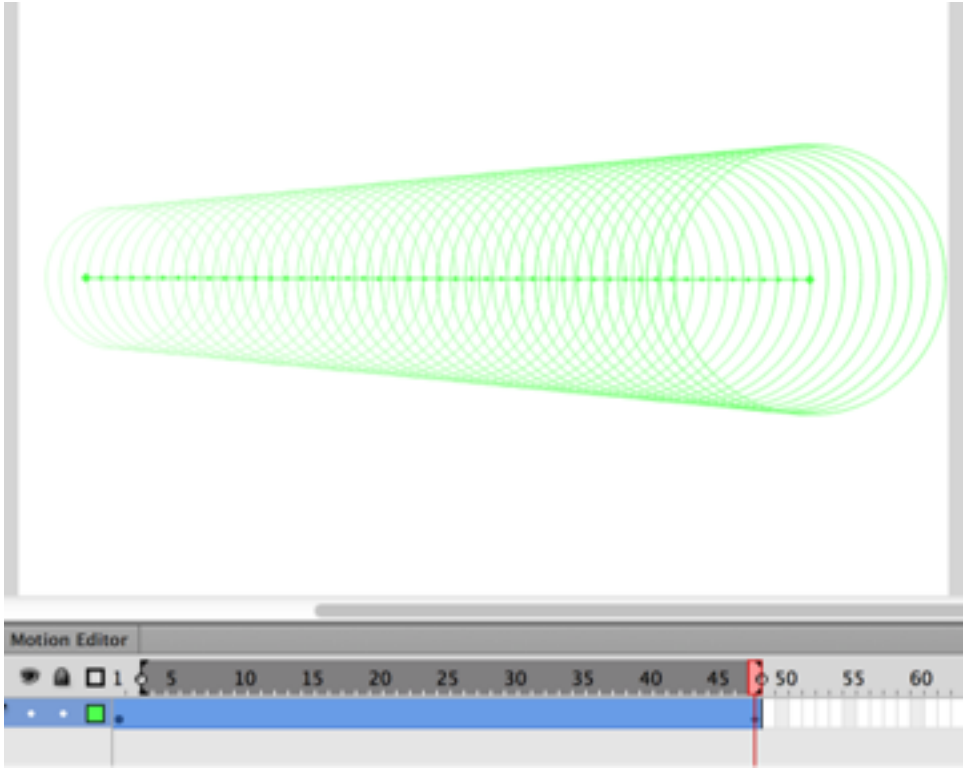

```
class SheetSprite(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.sheet = pygame.image.load('spritesheet.png').convert()
        (self.sheet_w, self.sheet_h) = self.sheet.get_size()
        self.posX = 0
        self.posY = 0
        self.image = self.sheet.subsurface((self.posX,self.posY,spr_w,spr_h)).copy()
        self.rect = self.image.get_rect()

    def update(self):
        nextImage = self.sheet.subsurface((self.posX,self.posY,spr_w,spr_h)).copy()
        self.image = nextImage
        self.posX += spr_w
        if self.posX >= self.sheet_w:
                self.posX = 0
                self.posY += spr_h
        if self.posY >= self.sheet_h:
                self.posY = 0
```

Viewports

- Viewport:
  - Graphical representation of a small part of a larger image
  - Often automatically scrolled and updated
- Examples in lecture:
  - Individual frames from sprite sheet
  - Magnified image
- Examples in games:
  - Main character surroundings
  - Viewport onto larger images of scene

Animation via Interpolation (Tweening)

- Computation of in-between frames
  - Positions of elements in key frames are given
  - Rules for interpolation are given
  - Intermediate images are computed
- Either on full frame level or (better) for individual objects in scene
- Well-known example: Adobe Flash "Tweens"
- Built into high-level frameworks
  - E.g. Animation as special case of Action in Cocos2d-x



Example from Adobe Flash

7 Programming with Animations

7.1 Animated Graphics: Principles and History

7.2 Types of Animation

7.3 Programming Animations: Interpolation 

7.4 Design of Animations

Literature:

W. McGugan 2007 (see above)

K. Peters: ActionScript 3.0 Animation - Making Things Move!
Friends of ED/Apress 2007

Interpolation (General)

- Given: (Finite) set of data points
- Computed: New data points such that a function exists which
  - has the given data points in its graph
  - is defined on a given input range
  - fulfills certain constraints
- Most simple case: Linear interpolation
  - Two data points given
  - Computes a linear function
- Multimedia interpolation:
  - Discrete inputs and values for all functions
  - Example: Interpolating horizontal position along x-axis

Discrete Linear Interpolation

- Given:
  - Number *n* of steps (e.g. animation frames)
  - Value in step 0: v_{start}
  - Value in step n: v_{end}
- Compute:
  - Value *v* for all intermediate steps *i* between 0 and *n*
- Traditional (Newton) interpolation formula:

$$v_i = v_{start} + \frac{v_{end} - v_{start}}{n} \cdot i$$

- Using a constant:

$$dv = \frac{v_{end} - v_{start}}{n}$$

$$v_i = v_{start} + dv \cdot i$$

```
i = 0
while True:
    if i <= n:
        print 'Step no', i, ': v=', v
        i += 1
        v = vstart + dv*i
    else:
        break
```

Linear Interpolation of Position

```
vstart = 40
xstart = 40
xend = 600
n = 80 #Number of steps
dx = (xend - xstart)/n

i = 0
x = xstart
y = 240

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    if i <= n:
        pygame.draw.rect(scr, white, Rect((0, 0),(scr_w, scr_h)))
        pygame.draw.circle(scr, red, (x, y), 40)
        i += 1
        x = xstart + dx*i

    pygame.display.update()
```

But:

Interpolated variable (x) can be computed differentially

Speed of animation depends on computing speed

AnimationBasics0.py

Beware of Rounding Problems!

```
vstart = 40
vend = 500
vdiff = vend - vstart
dv = vdiff/n # // in Python 3!
v = vstart
i = 0

while True:
    if i <= n:
        print 'Step no', i, ': v=', v
        i += 1
        v = vstart + dv*i
    else:
        break
```

```
Step no 80 : v= 440
```

```
vstart = 40
vend = 500
vdiff = float(vend - vstart)
dv = vdiff/n   # easier in Python 3!
v = vstart
i = 0

while True:
    if i <= n:
        print 'Step no', i, ': v=', v
        i += 1
        v = vstart + dv*i
    else:
        break
```

```
Step no 80 : v= 500.0
```

QUIZ: Why are the results different?

Interpolation using Fixed Frame Rate

```
xstart = 40
xend = 600
framerate = 30 #frames per second
n = 80 #Number of steps
dx = (xend - xstart)/n

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

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()

    if x+40 <= scr_w:
        pygame.draw.rect(scr, white, Rect((0, 0), (scr_w, scr_h)))
        pygame.draw.circle(scr, red, (x, y), 40)
        x += dx

    timepassed = clock.tick(framerate)
    pygame.display.update()
```

Clock.tick() documentation

If you pass the optional framerate argument the function will delay to keep the game running slower than the given ticks per second. This can be used to help limit the runtime speed of a game. By calling Clock.tick(40) once per frame, the program will never run at more than 40 frames per second.

Differential positioning simplifies code

Termination test now based on scene!

Speed of animation relative to frame rate

AnimationBasics1.py

Computation of Speed

- Assume a given frame rate *fr*
- Specifying speed of an object in absolute terms
  - [pixel/second]
- How is the relationship between:?
  - the relative delta per frame *delta* [px]
  - the absolute speed of the object *speed* [px/s]
  - the frame rate *fr* [1/s]

$$delta \cdot fr = speed$$

- Consequence:

$$delta = \frac{speed}{fr}$$

```
speed = 210 #pixels per second
dx = speed/framerate
```

Alternative: Compute *dx* within loop from *timepassed* * *speed*

QUIZ

- Look at the source code of the preceding example (AnimationBasics1)
- Do we actually need the variable *xend*? What is its purpose?