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
7.5 Game Physics
Eadweard Muybridge: Chronofotografie

• 1830 – 1904

J. Stuart Blackton: The Father of Animation

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

The Enchanted Drawing
©November 16, 1900
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

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

4 Programming with Animations

4.1 Animated Graphics: Principles and History
4.2 Types of Animation
4.3 Programming Animations: Interpolation
4.4 Design of Animations
4.5 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 (Flash)

• Properties of a (2D) object manipulated by motion tween:
  – Position (x and y)
  – Rotation (z)
  – Skew/Shear (Neigung)
  – Size
  – Color effects

• Basic idea of graphically creating a motion tween:
  – Place an object (instance!) on a separate layer
  – Invoke “Create Motion Tween” (context menu)
  – Re-adjust 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)
Example: Motion Tween in Flash (2)
Example: Tweening Colours in Flash
Example: Tweening Object Size in Flash
Example: Shape Tweening in Flash

Shape tweening interpolates between geometric 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
4 Programming with Animations

4.1 Animated Graphics: Principles and History
4.2 Types of Animation
4.3 Programming Animations: Interpolation
4.4 Design of Animations
4.5 Game Physics

Literature:

W. McGugan 2007 (see above)
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: $v_{\text{start}}$
  – Value in step $n$: $v_{\text{end}}$

• Compute:
  – Value $v$ for all intermediate steps $i$ between 0 and $n$

• Traditional (Newton) interpolation formula:

\[
v_i = v_{\text{start}} + \frac{v_{\text{end}} - v_{\text{start}}}{n} \cdot i
\]

• Using a constant:

\[
dv = \frac{v_{\text{end}} - v_{\text{start}}}{n}
\]

\[
v_i = v_{\text{start}} + dv \cdot i
\]
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
**Beware of Rounding Problems!**

| vstart = 40  | vstart = 40  |
| vend = 500   | vend = 500   |
| vdiff = vend - vstart | vdiff = float(vend - vstart) |
| dv = vdiff/n # // in Python 3! | dv = vdiff/n  # easier in Python 3! |
| v = vstart    | v = vstart   |
| i = 0         | 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  
Step no 80 : v = 500.0

**QUIZ:** Why are the results different?
Interpolation using Fixed Frame Rate

```python
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
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 $timepassed \times speed$
QUIZ

• Look at the source code of the preceding example (AnimationBasics2)
• Do we actually need the variable $x_{end}$? What is its purpose?
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

...  
steps = (xend - xstart)/dx  
dsize = 1.0/steps  
dcolor = 1.0/steps  
clock = pygame.time.Clock()  
x = xstart  
y = 240  
r = 40  
sizefactor = 1  
colorfactor = 0

while True:  
    for event in pygame.event.get(): ...
        xr = r*sizefactor  
    if x+xr <= scr_w:  
        pygame.draw.rect(scr, white, Rect((0, 0), (scr_w, scr_h)))  
        pygame.draw.circle(scr, blend_color(red, blue, colorfactor), (x, y), int(xr))  
        x += dx  
        sizefactor += dsize  
        colorfactor += dcolor  
    timepassed_secs = clock.tick(framerate) / 1000.0  
    pygame.display.update()
Animation Using Transitions in JavaFX

- **Transitions:**
  - Simple built-in animation framework in JavaFX, using implicit time container
  - Various types of transitions: Fade, Fill, Path, Rotate, Scale, Stroke, Translate
  - Parallel and sequential composition of transitions

```java
Circle c = new Circle(40, 240, 40);
c.setFill(Color.RED);
root.getChildren().add(c);

TranslateTransition tt = new TranslateTransition(Duration.millis(3000), c);
tt.setToX(520);
...
p
```
Parallel Transitions in JavaFX

```java
Circle c = new Circle(40, 240, 40);...
root.getChildren().add(c);

TranslateTransition tt = new TranslateTransition
    (Duration.millis(3000), c);
tt.setToX(520);

ScaleTransition st = new ScaleTransition
    (Duration.millis(3000), c);
st.setToX(2.0f);
st.setToY(2.0f);

FillTransition ft = new FillTransition
    (Duration.millis(3000), c);
ft.setToValue(Color.BLUE);

ParallelTransition trans = new ParallelTransition();
trans.getChildren().addAll(tt, st, ft);
...
primaryStage.show();
trans.play();
```
Path Transitions in JavaFX

```java
Path path = new Path();
path.getElements().add(new MoveTo(40, 240));
path.getElements().add(
    new ArcTo(280, 200, 0, 320, 440, false, false));
path.getElements().add(
    new ArcTo(280, 200, 0, 600, 240, false, false));
path.getElements().add(
    new ArcTo(280, 200, 0, 320, 40, false, false));
path.getElements().add(
    new ArcTo(280, 200, 0, 40, 240, false, false));
path.setStroke(Color.BLACK);
path.setStrokeWidth(0.5);
root.getChildren().add(path);

PathTransition pt = new PathTransition();
pt.setDuration(Duration.millis(4000));
pt.setPath(path);
pt.setNode(c);
pt.setInterpolator(Interpolator.LINEAR);
pt.setCycleCount(Timeline.INDEFINITE);
```
Non-Linear Interpolation, Easing

• Plain linear interpolation does not match physical reality
  – Inertia, acceleration, deceleration, overshooting
• Custom interpolators
  – Use different computation of intermediate values
  – In Java FX: Create subclass of Interpolator and implement the curve() function (mapping [0.0 … 1.0] to itself)
• Frequently used non-linear interpolators:
  – EaseIn, EaseOut, EaseBoth
  – Smooth acceleration and deceleration of movement
  – In some frameworks, acceleration factors are adjustable
  – JavaFX: EaseBoth is default for Transitions, acceleration factor is 0.2
  – CreateJS: Ease in and easy out plus additional effects (bounce)
Timelines and Keyframes

- **Timeline**:  
  - Playable time container containing *key frames*
- **Key frame**:  
  - Carries time stamp relative to time line  
  - Container for *key values*
- **Key value**:  
  - Pair of (property of a stage object, value for the property)
- Interpolation between key frames is automatically inferred
Timeline Animation in JavaFX

Circle c = new Circle(40, 40, 40);
...
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
KeyFrame endFrame = new KeyFrame(Duration.millis(3000),
        new KeyValue(c.centerXProperty(), 600),
        new KeyValue(c.centerYProperty(), 40));
timeline.getKeyFrames().add(endFrame);
KeyFrame middleFrame = new KeyFrame(Duration.millis(1500),
        new KeyValue(c.centerYProperty(), 440));
timeline.getKeyFrames().add(middleFrame);
...
timeline.play();

<table>
<thead>
<tr>
<th>0 ms</th>
<th>1500 ms</th>
<th>3000 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.CenterX = 40</td>
<td>c.CenterX = 40</td>
<td>c.CenterX = 600</td>
</tr>
<tr>
<td>c.CenterY = 40</td>
<td>c.CenterY = 440</td>
<td>c.CenterY = 40</td>
</tr>
</tbody>
</table>
QUIZ

• What do we see if we run the preceding program?
Regularly Timed Update Events

• Many programs update the stage at regular positions in time
  – Framerate dependent clock in Pygame programs
  – EnterFrame event in Adobe Flash
  – KeyFrame events in JavaFX

• Typical realisation:
  – Event handler, being called at regular intervals
  – Timeline concept can be used
  – JavaFX: Event at certain (timed) key frames
  – Flash: Event when entering a frame
Time Event Animation in JavaFX

```java
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
KeyFrame moveCircle = new KeyFrame(Duration.millis(20),
    new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            if (c.getTranslateX()+2*r > sc_w ||
                c.getTranslateX() < 0) {
                dx = -dx;
            }
            if (c.getTranslateY()+2*r > sc_h ||
                c.getTranslateY() < 0) {
                dy = -dy;
            }
            c.setTranslateX(c.getTranslateX()+dx);
            c.setTranslateY(c.getTranslateY()+dy);
        }
    });
timeline.getKeyFrames().add(moveCircle);
...
timeline.play();
```
Interpolation (Tweening) in CreateJS

```javascript
var circle = new createjs.Shape();
circle.graphics
  .beginFill("red")
  .drawCircle(0, 0, r);
circle.x = 40;
circle.y = 240;
stage.addChild(circle);

createjs.Tween.get(circle,{loop:true})
  .to({x:st_w-2*r, scaleX:2.0, scaleY:2.0}, 4000,
      createjs.Ease.bounceOut);

createjs.Ticker.setFPS(30);
createjs.Ticker.addEventListener("tick", stage);
```