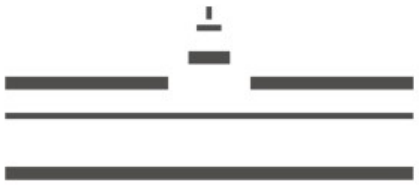


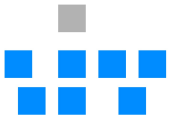
Westfälische
Wilhelms-Universität
Münster

Programmierung für mobile Endgeräte

Gestenerkennung & Animationen

wissen.leben
WWU Münster

Name: Patrick Förster



ZENTRUM FÜR
INFORMATIONS
VERARBEITUNG

Interaktion (Rückblick)

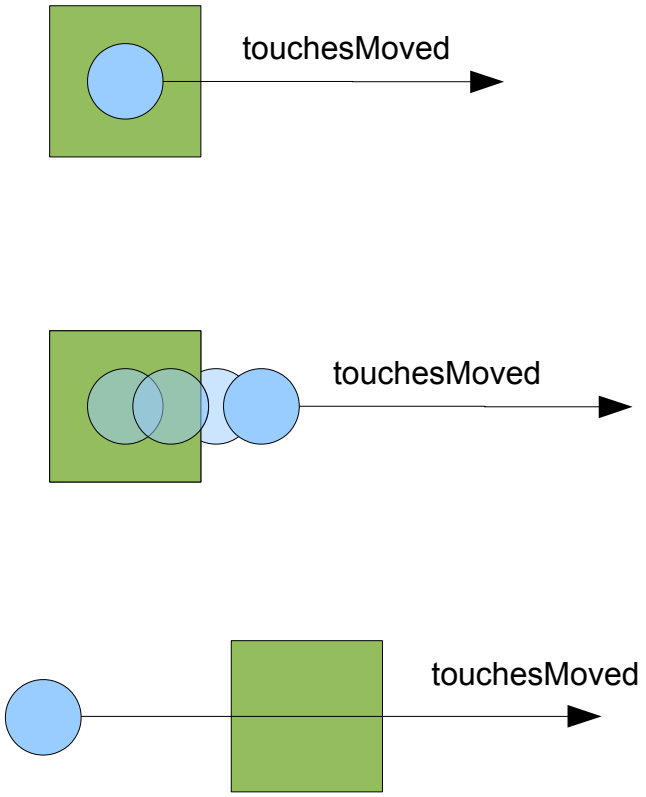
- Jede UIView-Klasse erbt von UIResponder und ist damit in der Lage Touch-Eingaben entgegen zu nehmen

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```

- Die Methode touchesCancelled wird immer dann aufgerufen, wenn das iOS entschieden hat, dass eine bereits begonnene Eingabe nicht mehr valide ist
- Das trifft bspw. ein wenn während einer Berührung die Home-Taste gedrückt wird, oder wenn die maximal Anzahl an Multi-Touch-Eingaben überschritten wird
- Per Default unterstützen UIView-Komponenten lediglich einfache Eingaben
- Multitouch-Unterstützung kann über den „Attribute Inspector“ („Interaction“) aktiviert werden
- Alle registrierten Berührungen werden im Parameter touches mitgeliefert
- Liegen zwei Objekte übereinander bekommt das obere den Zuschlag, während das untere leer ausgeht

Eingabe + Bewegung

- Falls eine Berührung gehalten und der Finger bewegt wird, erfolgt in bestimmten Abständen der Aufruf der touchesMoved Methode
- Die Methode wird auf einer Komponente auch dann aufgerufen, wenn der Finger im Laufe der Bewegung die Bounding-Box der Komponente verlässt
- Sie wird allerdings nur für solche Komponenten aufgerufen, die auch den initialen touchesBegan Aufruf mitbekommen haben
- Wenn also während einer Bewegung eine „neue“ Komponente berührt wird, nimmt diese nicht an der Bewegung teil



Einfaches Dragging

- Da nur solche Objekte ein Update per touchesMoved erhalten, die initial berührt wurden, lässt sich allein mit dieser Methode ein einfaches Drag-Verhalten implementieren

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch* touch = [touches anyObject];                                          (A)

    CGPoint point = [touch locationInView: self.superview];                        (B)
    point = CGPointMake(point.x - self.frame.size.width*0.5, point.y - self.frame.size.height*0.5);   (C)
    self.frame = CGRectMake(point.x, point.y, self.frame.size.width, self.frame.size.height);         (D)
}
```

- Für das Bewegen von Objekte ist eine einfache Eingabe vollkommen ausreichend, weshalb hier einfach die erst beste Eingabe als Referenz genommen wird (A)
- In (B) wird die Position der Eingabe relativ zum darüber liegenden View berechnet (hier wird angenommen, dass der „Superview“ gleichzeitig der Root-View ist, in dem die Bewegung stattfinden soll)
- In (C) wird das zu bewegende Objekt zentriert...
- ... und letztendlich in (D) auf die Position der Eingabe geschoben

Treffer-Erkennung

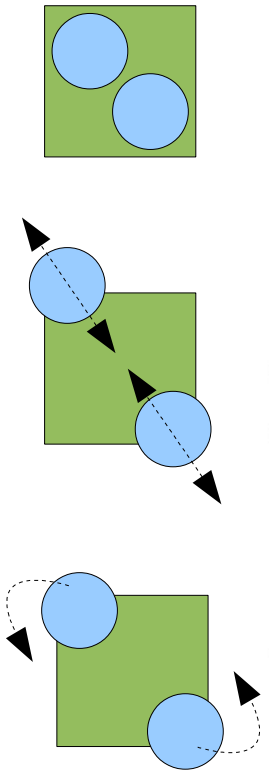
- Jedes UIView-Objekt verfügt über die Methode hitTest:withEvent, die überprüft ob eine Eingabe innerhalb der eigenen Bounding-Box liegt und wenn ja, welches Objekt den Zuschlag bekommt
- Möchte man bspw. beim Ziehen über den Bildschirm Objekte „einsammeln“, so kann man die Treffer-Erkennung in Verbindung mit der Implementierung auf der vorherigen Folie dazu nutzen, Bewegung an UI-Komponenten zu delegieren, die davon eigentlich nichts mitgekommen hätten:

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch* touch = [touches anyObject];
    for (UIView* view in self.subviews) {
        CGPoint point = [touch locationInView: view];                                    (A)
        if ([view hitTest:point withEvent:event]) [view touchesMoved: touches withEvent: event];   (B)
    }
}
```

- Hier wird die Eingabeposition nicht relativ zum Root-View, sondern zu den einzelnen UI-Komponenten errechnet (A)
- Und anhand dieser Berechnung überprüft, ob eine Komponente berührt wurde (B)

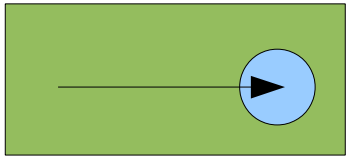
Neue Eingabemöglichkeiten

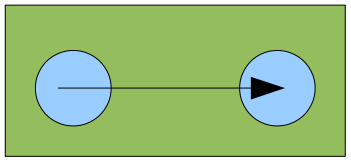
- Touch und Bewegung resultieren in einigen interessanten neuen Eingabemöglichkeiten:
  - **Tap**: Mit einem Tap ist das „klopfen“ auf einer bestimmten Komponente gemeint. Im einfachsten Fall ist dies das ein einfaches Tippen; man kann per Tap aber auch Doppel- bzw. Mehrfachklicks abbilden, die zudem vielleicht die Eingabe durch mehrere gleichzeitige Berührungen voraussetzen
  - **Pinch**: Mit einem Pinch, ist das Berühren und zusammen- bzw. auseinanderziehen von zwei Eingaben auf einer UI-Kompontente gemeint. Häufig wird diese Eingabe zur Umsetzung von Zoom-Operationen eingesetzt wird.
  - **Rotation**: Ähnlich wie Pinch benötigt eine Rotation zwei Eingaben, die in eine beliebige Richtung um einen bestimmten Winkel gedreht werden

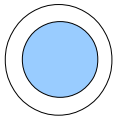


Neue Eingabemöglichkeiten

- **Swipe**: Mit Swipe bezeichnet man das Wischen über den Bildschirm. Diese Eingabe wird häufig zum Blättern innerhalb einer View-Hierarchie genutzt
- **Pan**: Panning bezeichnet das Ziehen einer UI-Komponente und kann analog zur bereits besprochenen Drag-Operation verstanden werden
- **Long Press**: Eine Long-Press Eingabe ist eine Eingabe auf einer UI-Komponente die über längere Zeit gehalten wird







- Wichtig: Pan und Swipe verfolgen beide die Bewegung einer Eingabe, um anhand dieser die Verschiebung zwischen zwei Punkte zu messen. Swipe ist also ein spezial Fall von Pan und es ist daher nicht möglich beide auseinander zu halten

Gestik in Xcode

- Alle beschriebenen Möglichkeiten besitzen im Cocoa-Touch Framework ein Pendant, das es ermöglicht ohne großen Programmieraufwand auf die Eingabe einer bestimmten Art zu reagieren
- Über das Storyboard lassen sich die von UIGestureRecognizer erbenden „Wächter“ auf jede beliebige Komponente anwenden
- Per IBAction kann ein „Wächter“ mit dem Controller verbunden werden und so auf bestimmte Gestiken reagiert werden:

```
- (IBAction)handleTap:(UITapGestureRecognizer *)recognizer {
    CGFloat r =  (rand() % 255 / 255.0f);
    CGFloat g =  (rand() % 255 / 255.0f);
    CGFloat b =  (rand() % 255 / 255.0f);

    self.backgroundColor = [UIColor colorWithRed:r green:g blue:b alpha:0.5f];
}
```

- Hier wird bei einem Tap die Hintegrundfarbe der Komponente auf eine per Zufall generiere Farbe gesetzt

Beispielanwendung

- Als Beispielanwendung für einen Großteil der vordefinierten Gestiken soll eine kleine Anwendung geschrieben werden, die der Oberfläche des iOS-Betriebssystem ähnelt
- Auf einer beliebigen Anzahl von Seiten, durch die per Swipe geblättert werden kann und die per Pinch gezoomt werden können, werden eine beliebige Anzahl von Kacheln dargestellt
- Per Long-Press auf eine Kachel kann die Ansicht in den Editmodus umgeschaltet werden, in dem es möglich ist, einzelne Kacheln per Tap zu löschen
- Für ein wenig Abwechselung soll das Blättern animiert sein
- Im Editmodus sollen sich die Kacheln zudem leicht bewegen

Oberfläche der App: Seite 1

Editmodus: Kacheln rotieren um einen bestimmten Winkel

Layout der Beispielanwendung

- Die Applikation basiert auf einem Single-View Projekt
- Für den View wird eine eigene Klasse definiert (PGTileView), die über ein Delegate verfügt, das im Storyboard mit dem autogenerierten ViewController verbunden wird
- Der Hauptview ist aufgeteilt in zwei Bereiche:
  - *Paging*: Im oberen Bereich wird die aktuelle Seite anzeigt. In diesem Bereich werden Zoom und Swipe Gestiken entgegen genommen
  - *Control*: Im unteren Bereich wird ein Button eingeblendet, über den der Editmodus (analog zum „Home“-Button) wieder beendet werden kann.
- Die Aufteilung ergibt sich daher, dass der Control-Button weder mitgezoomt noch -geblättert werden soll
- Für die Kacheln wird ebenfalls eine eigene Klasse angelegt (PGTile)
- Um die Kacheln zu kapseln und Operationen auf allen Kacheln einer bestimmten Seite gleichzeitig durchführen zu können, wird zudem noch die Klasse PGTilePage definiert

Registrieren des Pinch- und des Swipe- Wächters

- Sobald der Root-View geladen wurde, wird der Page-View erstellt und die nötigen Gestik-Wächter definiert:

```
UIPinchGestureRecognizer* pinchRecognizer =
    [[UIPinchGestureRecognizer alloc] initWithTarget:_delegate action:@selector(handlePinch:)];     (A)
[_pageView addGestureRecognizer:pinchRecognizer];

UISwipeGestureRecognizer* swipeRecognizer;
swipeRecognizer = [[UISwipeGestureRecognizer alloc]
    initWithTarget:_delegate action:@selector(handleSwipe:)];                                       (B)
[_pageView addGestureRecognizer:swipeRecognizer];
swipeRecognizer.direction = UISwipeGestureRecognizerDirectionLeft;
… // analog ein zweiter UISwipeGestureRecognizer für UISwipeGestureRecognizerDirectionRight
```

- Das Objektattribut delegate hält eine Referenz auf den ViewController, in dem sowohl die Methoden handlePinch: als auch handleSwipe: definiert sind
- Diese Methoden werden als Callback über den Initialisier dem jeweiligen Wächter zugeordnet (A, B)
- Bisher wurden diese Zuordnungen immer über das Storyboard und eine IBAction-Methode durchgeführt; wie man sieht, geht dies auch programmatisch

Pinch

- Mit der Pinch-Eingabe soll die aktuelle Seite gezoomt werden können
- Dazu sei angenommen, dass der ViewController Zugriff auf das zur Seite gehörende UIView-Objekt hat, bspw. über einen Index:

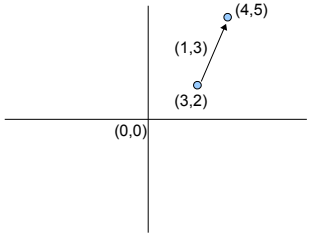
```
PGTilePage* page = [view getPageWithIndex:_page];
```

- Die Pinch-Eingabe wird so umgesetzt, dass nach dem Beenden des Zooms, das View-Objekt wieder in seinen Ursprungszustand zurückgesetzt wird
- Dazu muss feststellbar sein, wann eine Gestik beendet wurde
- Jede UIGestureRecognizer-Instanz liefert über state den Status der Eingabe
- Sobald eine Eingabe beendet wurde, wird die Callback-Methode noch ein weiteres Mal mit state == UIGestureRecognizerStateEnded aufgerufen

```
- (IBAction)handlePinch:(UIPinchGestureRecognizer *)recognizer {
    if (recognizer.state == UIGestureRecognizerStateEnded) {
        //
    }
    else {
        //
    }
}
```

Pinch (II)

- Grafische Operationen wie Skalieren (Zoom), Bewegen oder auch Rotieren werden in der Regel durch affine Transformation repräsentiert, die durch Matrixoperationen auf homogenen Koordinaten durchgeführt werden



$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+t_x \\ y+t_y \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3+1 \\ 2+3 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 5 \\ 1 \end{pmatrix}$$

Translation eines Punktes im 2d-Raum:

- Jede UI-Komponente verfügt über die Möglichkeit affine Transformationen auf ihr durchzuführen, so z.B. Skalierung für die Umsetzung des Pinch-Eingabe:

```
if (recognizer.state == UIGestureRecognizerStateEnded) page.transform = CGAffineTransformIdentity;
else {                                                    //          x-skalierung        y-skalierung
    page.transform = CGAffineTransformScale(page.transform, recognizer.scale, recognizer.scale);
    recognizer.scale = 1;    // Zurücksetzen, ansonsten würde sich der Faktor mit jeder Pinch-Bewegung akkumulieren
}
```

- Ist die Eingabe beendet worden, wird die Transformationsmatrix auf die Identität zurückgesetzt und so der Ursprungszustand wiederhergestellt
- Ansonsten wird der durch den Wächter gelieferte Skalierungsfaktor für die Skalierung in X- und Y-Richtung eingesetzt

Swipe

- Für die Umsetzung der Swipe-Eingabe werden alle Seiten in die Richtung des Swipes bewegt
- Da die einzelnen Seiten den gleichen Abstand zueinander haben, kann eine komplette Seite „geblättert“ werden, in dem sie um die Breite des Root-Views in X-Richtung verschoben wird; die Y-Koordinate bleibt beibehalten

```
// Controller
-(IBAction) handleSwipe:(UISwipeGestureRecognizer*)recognizer {
    switch (recognizer.direction) {
        case UISwipeGestureRecognizerDirectionLeft: [view shiftX:-1*view.frame.size.width]; break;
        case UISwipeGestureRecognizerDirectionRight: [view shiftX:view.frame.size.width]; break;
    }
}
// View
-(void) shiftX: (CGFloat) x {
    for (UIView* temp in _pageView.subviews)  temp.center = CGPointMake(temp.center.x + x, temp.center.y);
}
```

- Man könnte zum Verschieben auch eine affine Transformation einsetzen
- Das Verschieben sollte zudem nur durchgeführt werden, wenn noch Seiten in der gewünschten Richtung vorhanden sind

Long-Press

- Bei einer Long-Press-Eingabe wird in den Editmodus umgeschaltet, in dem einzelne Kacheln gelöscht werden können
- Dazu wird jeder Kachel ein Lösch-Button hinzugefügt, der in einer Xib-Datei definiert wurde und als Reaktion auf eine IBAction im View-Controller verweist

```
// Controller
- (IBAction)handleLongPress:(UIPanGestureRecognizer *) recognizer {
    if (self.editing) return ;
    self.editing = YES;
    PGTileView* view = ((PGTileView*)self.view);
    [view setEditing:YES];    // iteriert durch jede Seite und ruft „toggleEdit“ auf jeder Kachel auf
}
// PGTile
-(void) toggleEdit {
        ...
        id delegate = [self getPage].delegate;
        UIButton* button = [PGUIFactory createViewFromXib:@"uitiles" widthTag:PG_TILE_DELETE andFrame:
                                        CGRectMake(-8, -8, 29, 29); forOwner: delegate];
        [button addTarget:delegate action:@selector(onDelete:) forControlEvents:UIControlEventTouchDown];
        [self addSubview:button];
}
```

Objective-C: Blocks

- Die Eingaben sind damit alle verarbeitet; es bleibt die einzelnen Operationen, wie bspw. das Löschen einer Kachel logisch umzusetzen (hier ausgespart)
- Was bisher noch nicht umgesetzt wurde, sind die gewünschten Animationen
- Dazu muss ein neues Objective-C Konstrukt eingeführt werden: Blöcke
  - Blöcke sind eine Möglichkeit „anonyme“ Codefragmente zu definieren, die wie normale Datentypen genutzt werden können
  - Insbesondere können Blöcke als Objekt-Attribute oder Methoden-Parameter genutzt werden, um bestimmten Code zu einem anderen Zeitpunkt als den der Deklaration auszuführen
  - Blöcke haben einen einen eindeutigen Typ und können wie Funktionen Werte zurückliefern
  - Blöcke kapseln den Scope in dem sie deklariert wurden, nicht den, in dem sie ausgeführt werden!
  - Blöcke können Werte in dem Scope, in dem sie deklariert wurden, ändern
  - Scope-Werte, die von einem Block genutzt werden, überleben den Scope

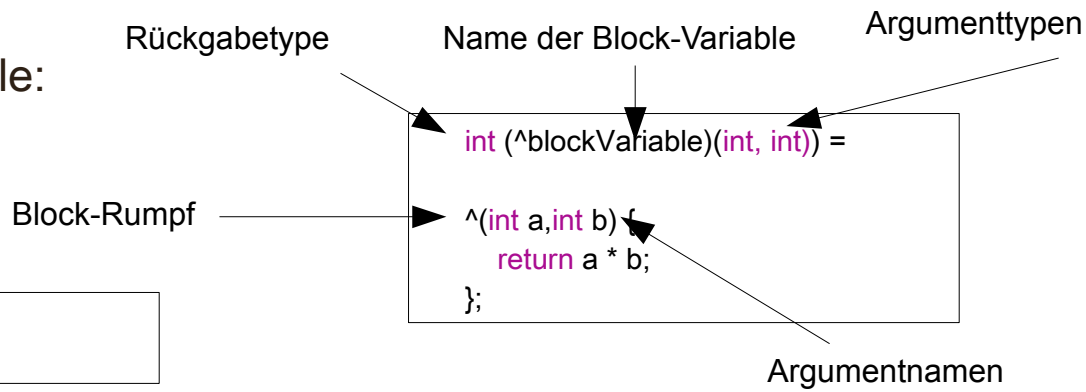
Objective-C: Blocks (II)

- Deklaration einer Block-Variable:
- Aufruf eines Block:

```
int result = blockVariable(17, 37);
```

- Definition eins Blocktyps:

```
typedef int (^Block)(int, int);
Block block = ^(int a, int b) {return a*b;};
```



- Blöcke speichern den Kontext (Scope) in dem sie deklariert wurden:

```
typedef NSString* (^Block)(void);
@implementation BlockFactory
-(Block) createBlock: (NSString*) name {
    return ^(void) {
        return [NSString stringWithFormat:@"My name is: %@. I
was produced by %@", name, self.description];
    };
}
@end
```

```
@implementation PGBlockDemo

-(void) run {
    BlockFactory* factory = [BlockFactory new];
    Block block = [factory createBlock: @"Block Nr. 1"];
    NSLog(@"%@", block());
}

@end
```

- Ausgabe: `My name is: Block Nr. 1. I was produced by <BlockFactory:...>`
- Das self in der Definition referenziert die Fabrik und nicht das self im Aufrufkontext

UI-Animation

- Die Klasse UIView bietet mit der Methode animateWithDuration: delay: options: animations: completion eine einfache Möglichkeit Animationen für eine UI-Komponente zu definieren, ohne dass man selbst das Handling für die Animation übernehmen muss
- Die beiden Parameter animations und completion erwarten dabei jeweils ein Block-Objekt, um die Animation an sich und das Verhalten nach Ablauf zu definieren
- Über das options-Attribut können per Bit-Flag Optionen für die Animation definiert werden, während delay festlegt, wie lange in Sekunden gewartet werden soll, bis die Animation startet und der erste Methodenparameter besagt, wie lange die Animation dauern soll

UI-Animation: Swipe

- Bei einem Swipe wechseln die Seiten bisher direkt ohne Animation, was einen sehr statischen Eindruck hinterlässt
- Mit animateWithDuration kann sehr leicht ein „weiches“ Blättern zwischen den Seiten realisiert werden:

```
// PGTilePage
-(void) animateShift: (CGFloat) shift onCurrenctPage: (NSInteger) page inDirection: (UISwipe...) direction {
    [UIView animateWithDuration:.25 delay:0 options:0
                animations:^{[self shiftX:((direction == UISwipeGestureRecognizerDirectionLeft) ? -1 : 1) * shift];}
                completion:nil
     ];
}
```

- Wichtig: Die Definition des Blocks befindet sich innerhalb der PGTilePage-Klasse, so dass self auch wirklich auf eine bestimmte Seite verweist!
- Im animations-Block wird per bekannter Shift-Methode die Seite verschoben
- Der Trick hierbei ist, dass die Komponente nicht direkt verschoben wird, sondern die Animation die Verschiebung linear auf die angegebene Zeit verteilt, so dass die Verschiebung als Animation sichtbar wird

UI-Animation: Long-Press

- Nachdem die Ansicht in den Editmodus geschaltet wurde, sollen die Kacheln sich bewegen, indem sie leicht vorwärts-rückwärts rotiert werden

```
// PGTile
-(void) animateEdit {
   [UIView animateWithDuration:.25 delay:0 options:0
                animations: ^(void) {self.transform = CGAffineTransformRotate(self.transform, _rFactor);}
                completion:^(BOOL finished) { _rFactor *= -1; [self animateEdit]; }
    ];
}
```

- Die Rotation wird durch das Objekt-Attribut (!) bestimmt
- Analog zur Verschiebung wird die Rotation linear auf die Zeit verteilt
- Sobald das Rotationsende erreicht wurde, wird der Faktor invertiert und per rekursiven Aufruf die Animation erneut gestartet
- Durch den rekursiven Aufruf entsteht eine Endlos-Animation
- Die gleiche Animation ohne Rekursion bekommt man mit den mit der Optionskombination UIViewAnimationOptionAutoreverse | UIViewAnimationOptionRepeat