Core Animation

Andreas Monitzer, 2010-12-09

back in black

Was ist Core Animation?

- Teil des QuartzCore-Frameworks
- OpenGL-Abstraktion
- behandelt Animationen
- Ursprung vermutlich CoverFlow

Konzept

- Layers
  - CALayer
- Animationen
  - CAAnimation

Anwendungen

- Coverflow
- Frontrow
- “moderne” views
- Spiele

CALayer

- ein Rechteck am Bildschirm
- Bild als Inhalt (dynamisch oder statisch)
- Alpha-Blending
- 3D-Transformationen

Basics

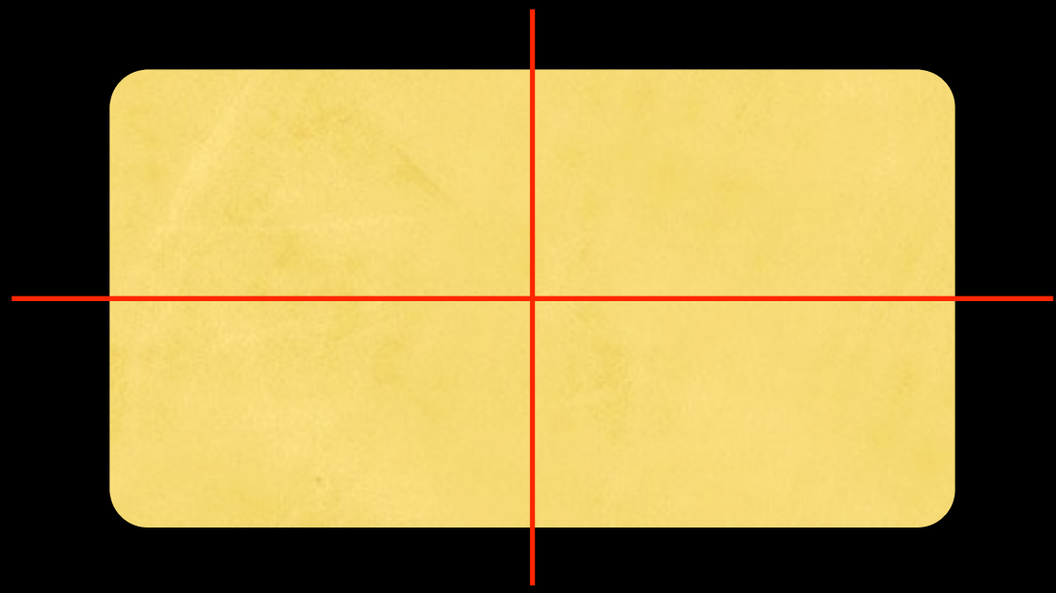
- Initialisation:
  - CALayer *layer = [CALayer layer];
- z.B. Farbe setzen:
  - layer.backgroundColor = CGColorGetConstantColor (kCGColorWhite);

Properties

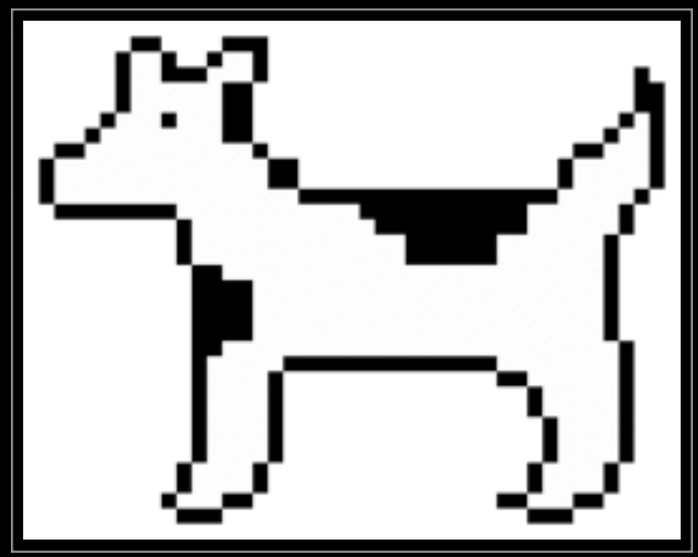
- bounds
- position
- zPosition
- transform
- opacity
- cornerRadius
- borderWidth
- borderColor
- shadow…
- filters (Mac only)
- sublayers
- contents (CGImageRef)

Anchor

- Welcher Punkt des Layers soll der Ursprung sein?
- layer.anchor = CGPointMake(…, …);
- [0, 1]
- default 0.5, 0.5

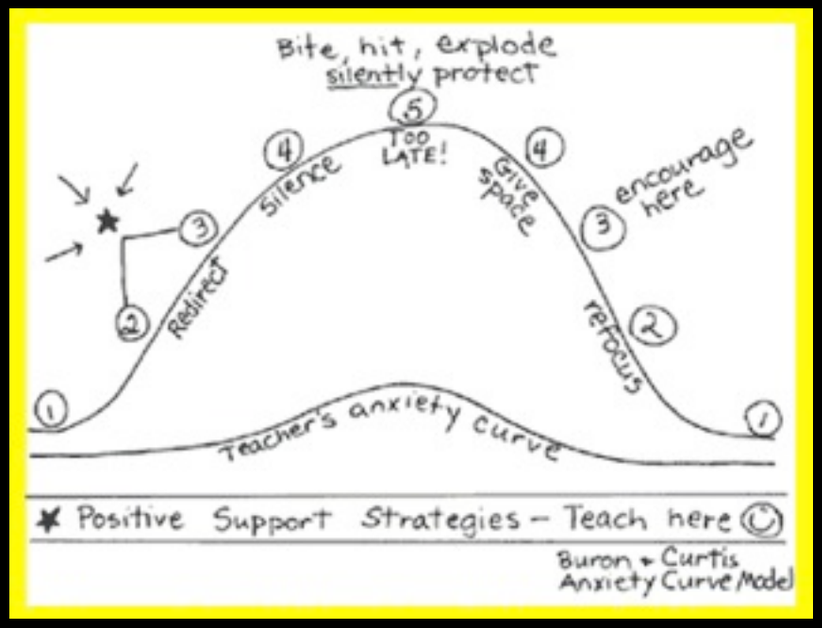


Inhalt



- einfach ein Bild setzen:
  - layer.contents = (id)myCGImage;
- oder manuell zeichnen

Manuelles Zeichnen



- Delegate-Option
  - - (void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx
- Subklasse von CALayer
  - - (void)drawInContext:(CGContextRef)ctx

Zeichenbefehle

- CoreGraphics, CGContext*
- z.B. CGContextFillRect
- Pfade
  - Kombination aus mehreren Zeichenbefehlen
  - erlauben beliebige gefüllte Flächen zu zeichnen (even-odd-rule)

Cocoa-Zeichenbefehle

- z.B. für Text, NSDrawNinePartImage und alten Zeichencode
- Core Animation Cookbook → Drawing

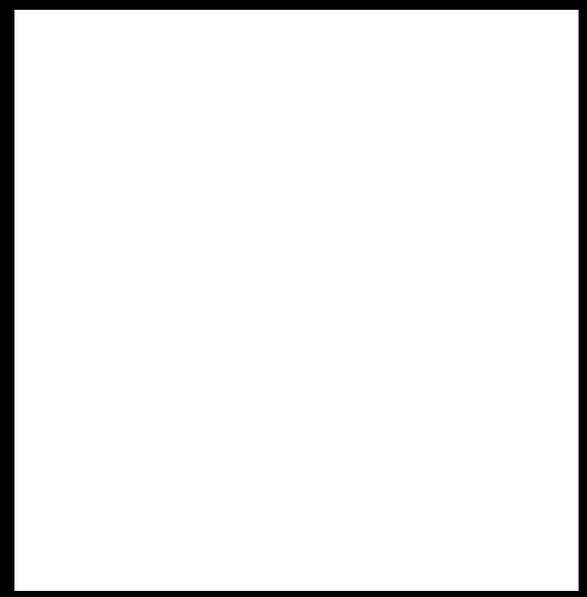
```
- (void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx {
    NSGraphicsContext *nsGraphicsContext;
    nsGraphicsContext = [NSGraphicsContext graphicsContextWithGraphicsPort:ctx flipped:NO];
    [NSGraphicsContext saveGraphicsState];
    [NSGraphicsContext setCurrentContext:nsGraphicsContext];

    // ...Draw content using NS APIs...

    [NSGraphicsContext restoreGraphicsState];
}
```

ACHTUNG

- layer.needsDisplayOnBoundsChange = YES;
  - sonst passiert gar nix

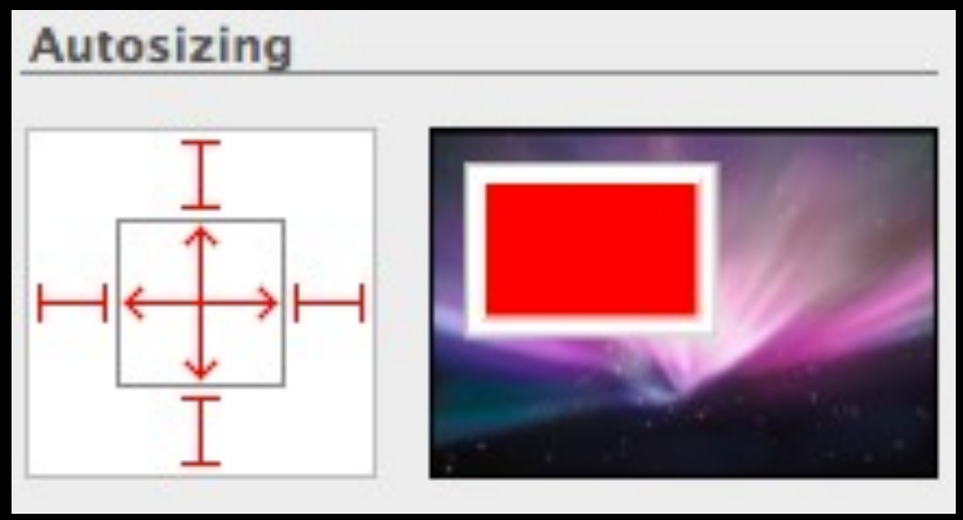


Minimum

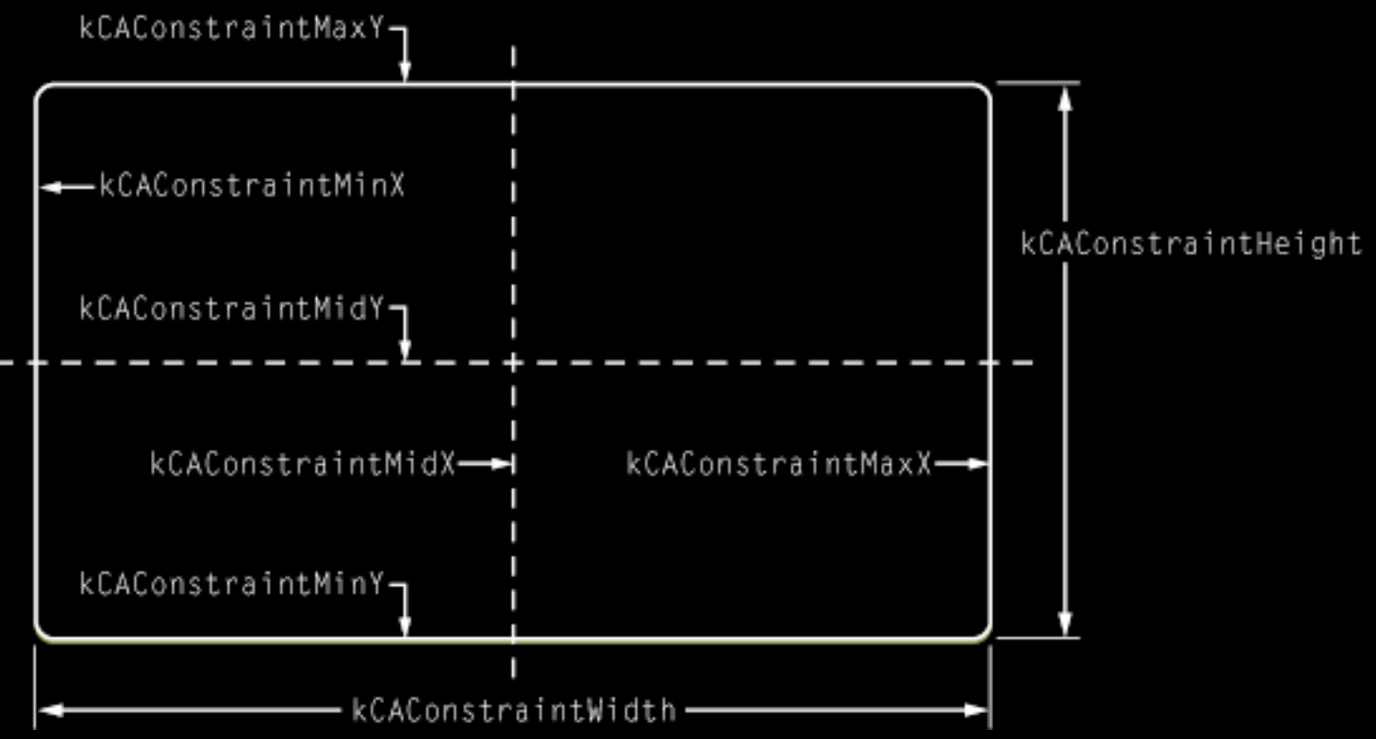
```
CALayer *layer = [CALayer layer];
layer.bounds = …;
layer.position = …;
layer.backgroundColor = …;
[upperlayer addSublayer:layer];
```

Resizing

- Springs&Struts wie bei views, zB.:
- layer.autoresizingMask =
  kCALayerWidthSizable |
  kCALayerHeightSizable;



Resizing, Variante 2



- Mac only!
- CALayoutManager > CAConstraintsLayoutManager
- layer.layoutManager =
  [CAConstraintLayoutManager layoutManager];
- “Das horizontale Zentrum dieses Layers ist 40%+20pt vom rechten Rand dieses anderen Layers.”

Beispiel

```
layer.constraints = [NSArray arrayWithObjects:
    [CAConstraint constraintWithAttribute:kCAConstraintMidX
relativeTo:@"superlayer" attribute:kCAConstraintMinX offset:location.x],
    [CAConstraint constraintWithAttribute:kCAConstraintMinY
relativeTo:@"superlayer" attribute:kCAConstraintMinY offset:location.y+10.0],
    nil];
```

Resizing, Variante 3

- Subklasse

  - (void)resizeWithOldSuperlayerSize:(CGSize)size

Integration mit Views

- Mac: [view setWantsLayer:YES];
  - IB-Support da, aber defekt
- iOS: immer aktiv
  - + (Class)[UIView layerClass]
- view.layer = …;
- oder als Sublayer hinzufügen

Events

- Layer haben kein event handling!
- Der view kann aber hit detection machen:
  - - (CALayer *)[CALayer hitTest: (CGPoint)thePoint]
  - Aufpassen wegen Animationen!

NSView-Transformationen

- -setWantsLayer: aktiviert OpenGL-rendering aller Cocoa-Controls in diesem View
- Veränderungen an [view layer] betreffen auch Controls
  - Event tracking bekommt davon aber nichts mit!

Animationen

- z.B. Verschieben eines Layers, Veränderung der Farbe, Überblendung zwischen zwei Bildern, …
- Implizit und explizit

Implizite Animationen

- passieren automatisch
  - z.B. layer.position = CGPointMake(0,0);
    - alle Animationen in einem runloop-Durchlauf passieren gleichzeitig
- konfigurierbar via CATransaction, z.B.
  - + (void)setAnimationDuration: (CFTimeInterval)duration

Temporär deaktivieren

```
[CATransaction begin];
[CATransaction setValue:(id)kCFBooleanTrue
    forKey:kCATransactionDisableActions];

…

[CATransaction commit];
```

Permanent deaktivieren

- Subklasse

```
- (id<CAAction>)actionForKey:(NSString *)event {
    return nil;
}
```

- alternativ im delegate (actionForLayer:forKey:)

Explizite Animationen

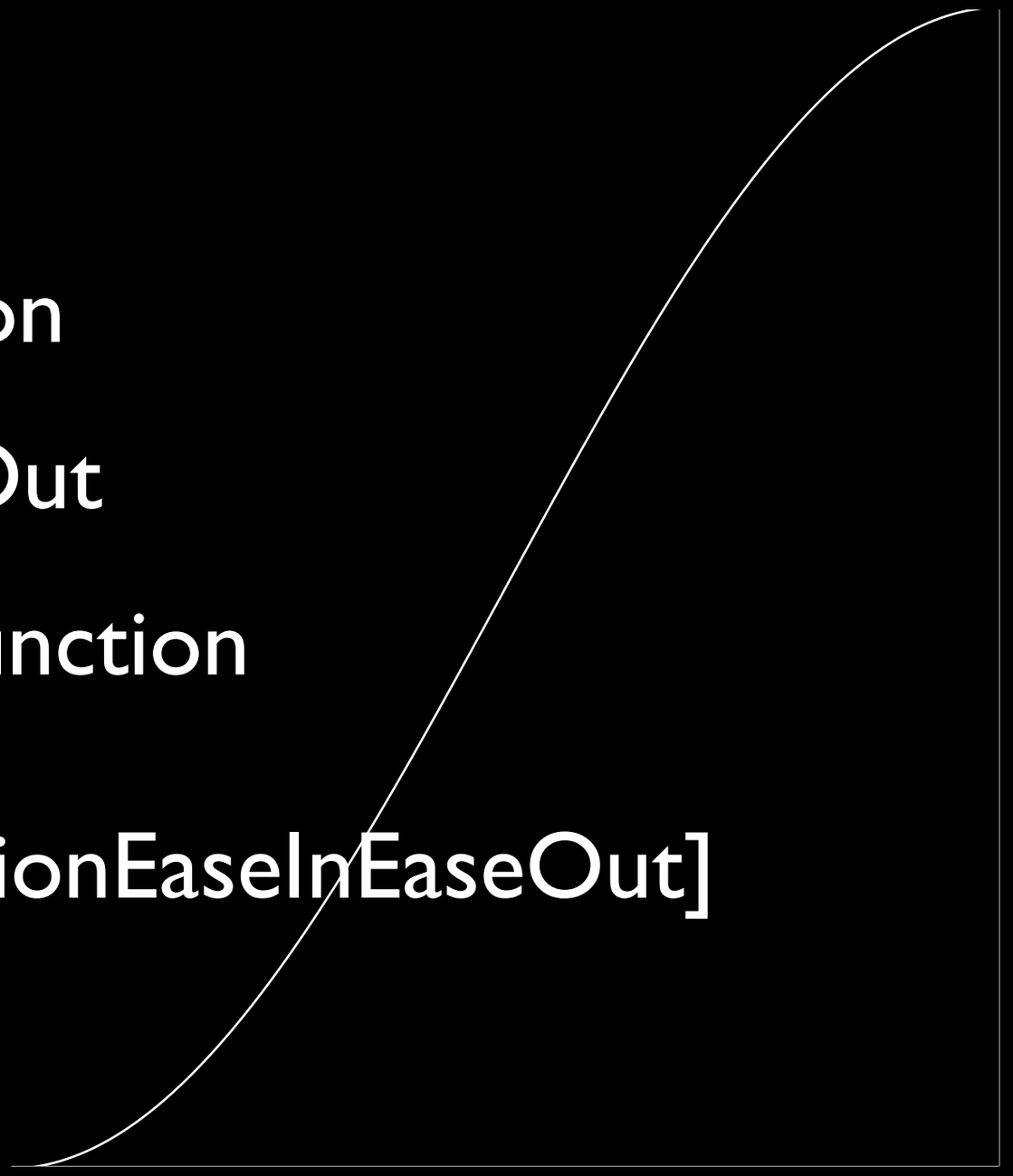
- z.B. für periodische Animationen
- ein Animationsobjekt, Subklasse von CAAnimation
- [layer addAnimation: forKey:];
  - startet automatisch
- Delegatefunktionen!

CABasicAnimation

- Key, der animiert werden soll
- From-Wert (NSValue: Zahl, Vektor, Farbe, Bild, etc)
- To-Wert
- Dauer
- wie oft (bis hin zu HUGE_VALF)
- autoreverse?
- Animationskurve

Animationskurven

- CAMediaTimingFunction
  - Linear/Ease In/Ease Out
- z.B. [CAMediaTimingFunction functionWithName: kCAMediaTimingFunctionEaseInEaseOut]



Custom properties animieren

```
+ (BOOL)needsDisplayForKey:(NSString *)key {
    if([key isEqualToString:@"phase"])
        return YES;
    return [super needsDisplayForKey:key];
}
```

Model und Presentation

- animierte Werte ändern sich nicht im layer selbst
- Es wird eine Kopie angefertigt, der “presentation layer”. Dieser wird gezeichnet.
  - -[CALayer modelLayer]
  - -[CALayer presentationLayer]

Auswirkungen

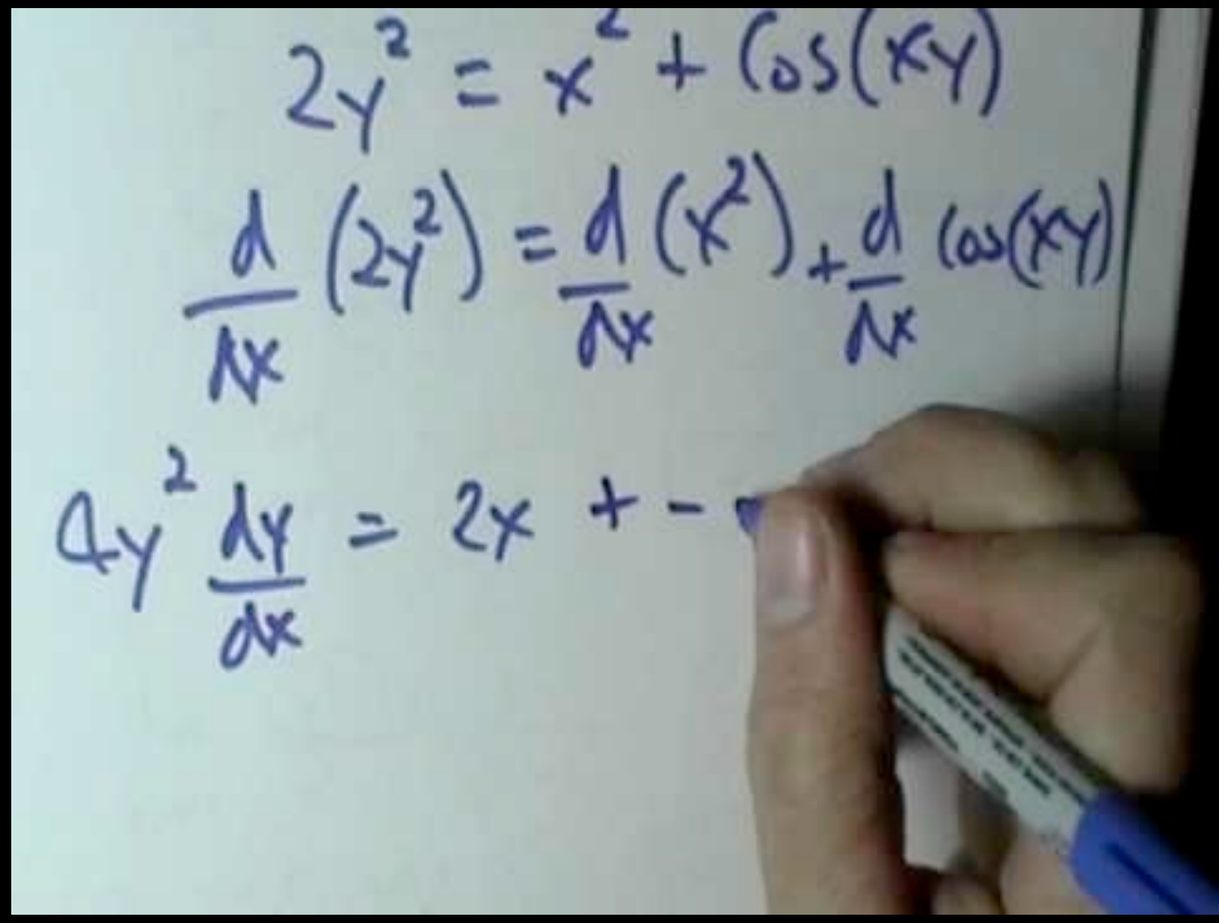
- custom properties werden nicht übernommen bei der Kopie, kann man selber kopieren
- Wert im model layer wird nicht geändert
  - “springt” zum alten Zustand nach der Animation per default
- hit detection in den presentation layers!

3D?

Naja, mehr oder weniger…

Ausflug in die Computergrafik

- Transformationen erlauben die Verschiebung von Punkten von einen Vektorraum in einen anderen
- 3 Vektorraum-Basisoperationen: Verschieben, Skalieren, Rotieren



Basiselemente

- Punkte im Raum sind dreidimensionale Vektoren

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Skalierung und Rotation können mit einer quadratischen Matrix mit der Dimension des Vektorraums dargestellt werden

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \qquad S(s) = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix}$$

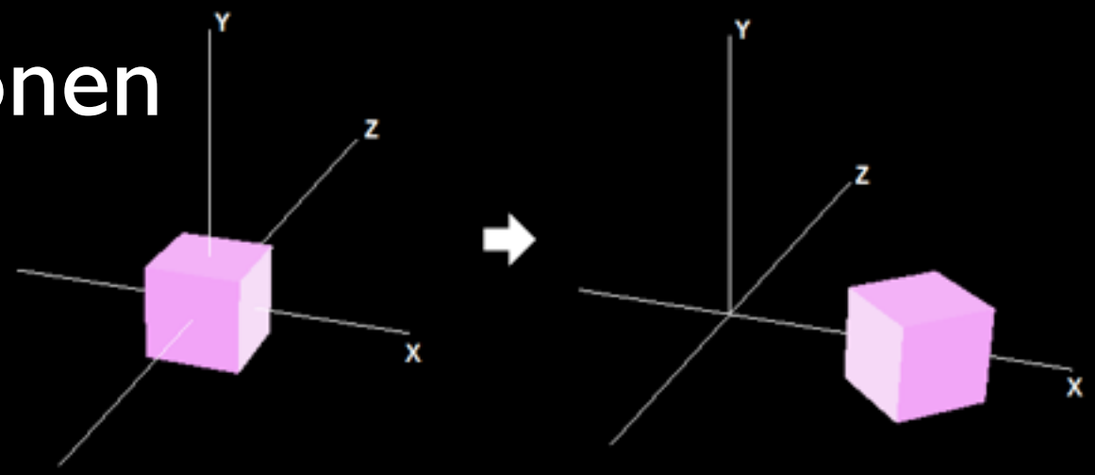
Anwendung

- Um eine Transformation auf einen Punkt anzuwenden, muss man multiplizieren:

$$P_s = \begin{pmatrix} x & y & z \end{pmatrix} \cdot \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix} = \begin{pmatrix} x \cdot s & y \cdot s & z \cdot s \end{pmatrix}$$

Verbindung

- Um mehrere Transformationen anzuwenden, muss man mehrere Multiplikationen machen.



$$P_{s2} = \begin{pmatrix} x & y & z \end{pmatrix} \cdot \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & s_1 \end{pmatrix} \cdot \begin{pmatrix} s_2 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_2 \end{pmatrix}$$

$$= \begin{pmatrix} x & y & z \end{pmatrix} \cdot \left(\begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & s_1 \end{pmatrix} \cdot \begin{pmatrix} s_2 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_2 \end{pmatrix} \right)$$

Translation

- Verschiebung nicht darstellbar!

$$P_t = \begin{pmatrix} x & y & z \end{pmatrix} \cdot ? = \begin{pmatrix} x + t_x & y + t_y & z + t_z \end{pmatrix}$$

Homogene Koordinaten

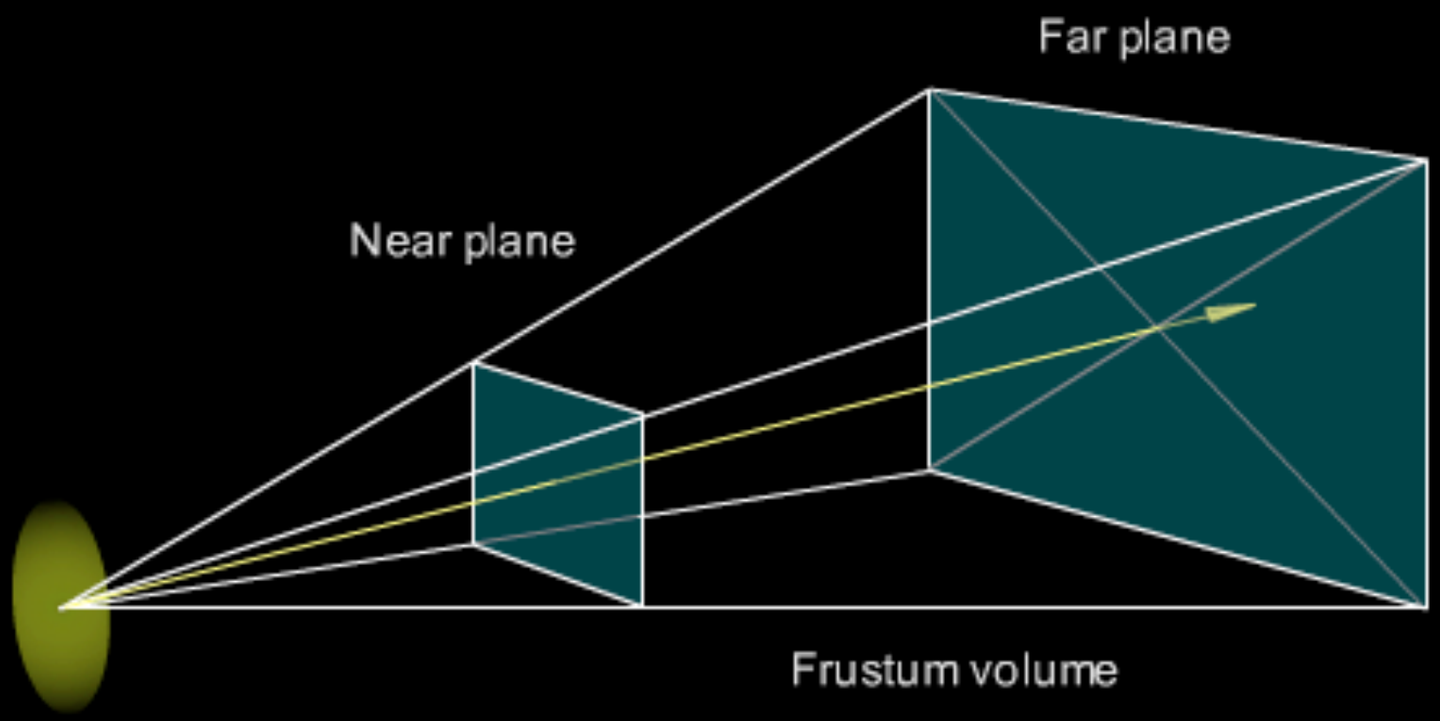
- Lösung: eine vierte Dimension (w)

$$P_t = \begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}$$

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

Perspektive

- Perspektive lässt Objekte, die weiter weg sind, kleiner erscheinen



Perspektivische Transformation

- 2 Schritte:
  - Transformation

$$P_p = \begin{pmatrix} x & y & z & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & p \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} x & y & z & p \cdot z \end{pmatrix}$$

  - Division

$$P_v = P_{p}./P_{pw}$$

$$= \begin{pmatrix} x & y & z & p \cdot z \end{pmatrix}./(p \cdot z)$$

$$= \begin{pmatrix} \frac{x}{p \cdot z} & \frac{y}{p \cdot z} & \frac{1}{p} & 1 \end{pmatrix}$$

Zurück zu Core Animation

- CATransform3D: 4x4-CGFloat-Matrix (struct)

- Generatoren
  - CATransform3DMakeScale
  - CATransform3DMakeTranslation
  - CATransform3DMakeRotation
- Konkatenatoren
  - CATransform3DScale
  - CATransform3DTranslate
  - CATransform3DRotate

Perspektive in Core Animation

- Achtung: Kein automatisches depth sorting!
  - "depth violation" sehr unangenehm

```
CATransform3D transform = CATransform3DMakeScale
(frame.size.width / kScaleFactor, frame.size.width / kScaleFactor, 1.0f);
transform.m34 = -0.00005f;
layer.sublayerTransform = CATransform3DTranslate
(CATransform3DRotate(transform, M_PI / 32.0f, 1.0f, 0.0f, 0.0f),
frame.size.width * 0.4f, frame.size.height / 2.0f + 100.0f, zPos);
```

- Experimentieren ist angesagt!

Was gibts noch?

- CAOpenGLLayer/CAOpenGLESLayer
- CAEmitterLayer (10.6)
- CATransformLayer (10.6)
- CATextLayer (Mac)
- QCCompositionLayer (Mac)
- QTMovieLayer (Mac)
- …

Beispiele