

Macoun' I I

Auflösungsunabhängiges Zeichnen

Frank Illenberger

Motivation

- „Houston, haben wir ein Problem?“
- Bewusstsein schärfen
- Einige Lösungen

Zeichnen

Zeichnen

- AppKit / UIKit
- OpenGL
- Quartz

Quartz-Zeichenmodell

- Bitmap-Bilder
- Linien-basiertes Zeichnen
- Text

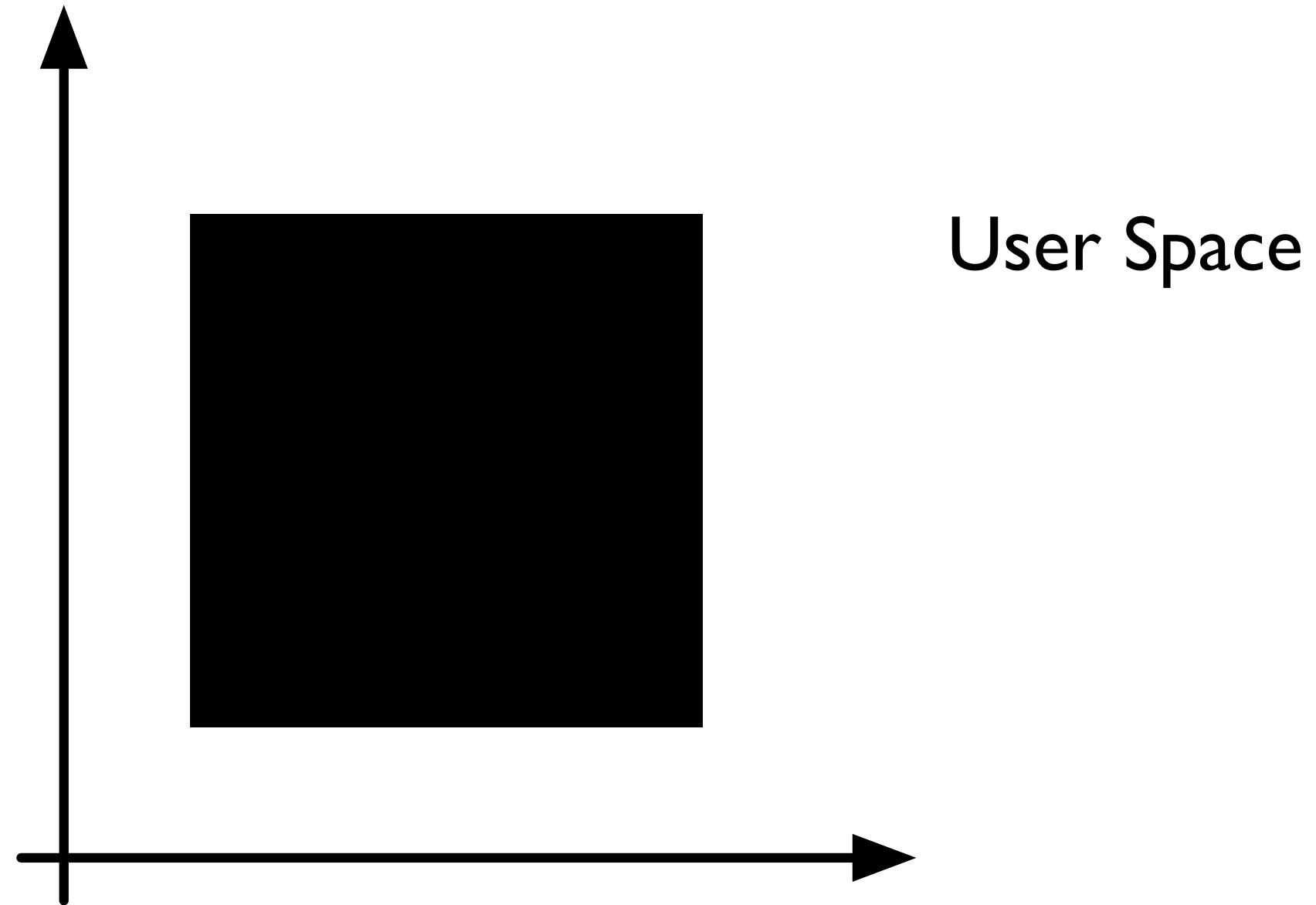
Koordinatensysteme

- „User Space“
 - abstraktes Koordinatensystem
 - keine Pixel – “Points” – Fließkomma-Genauigkeit
 - geräteunabhängig: Drucker, Bildschirme
- „Device Space“
 - geräteabhängig
 - meist verborgen
 - Fließkomma-Genauigkeit – wird auf entsprechenden Geräten final gerastert

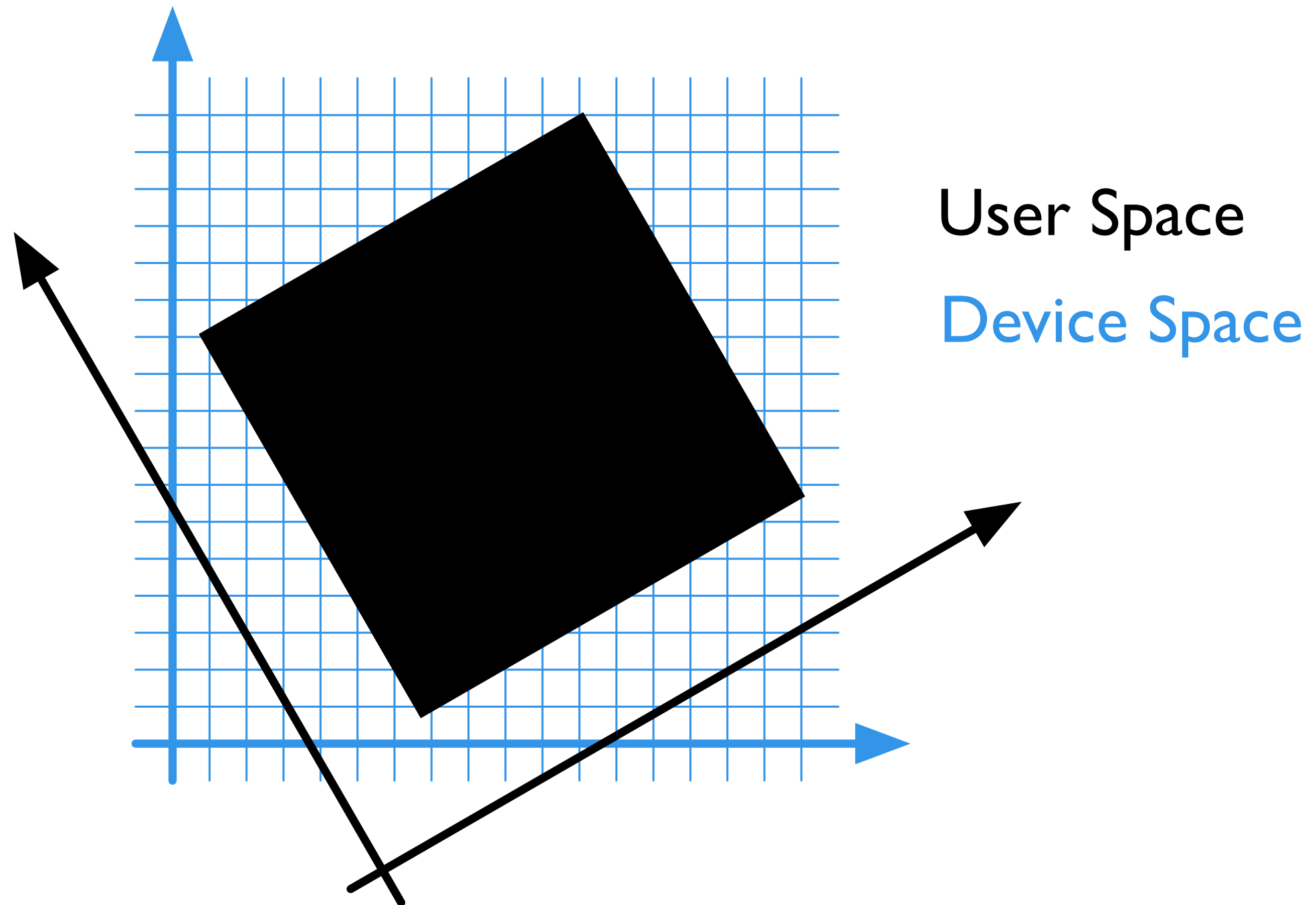
Koordinatensysteme

- Affine Transformation
 - Abbildung zwischen User- und Device Space
 - Kombination aus Verschiebung, Rotation und Skalierung
 - Auch in User Space schachtelbar

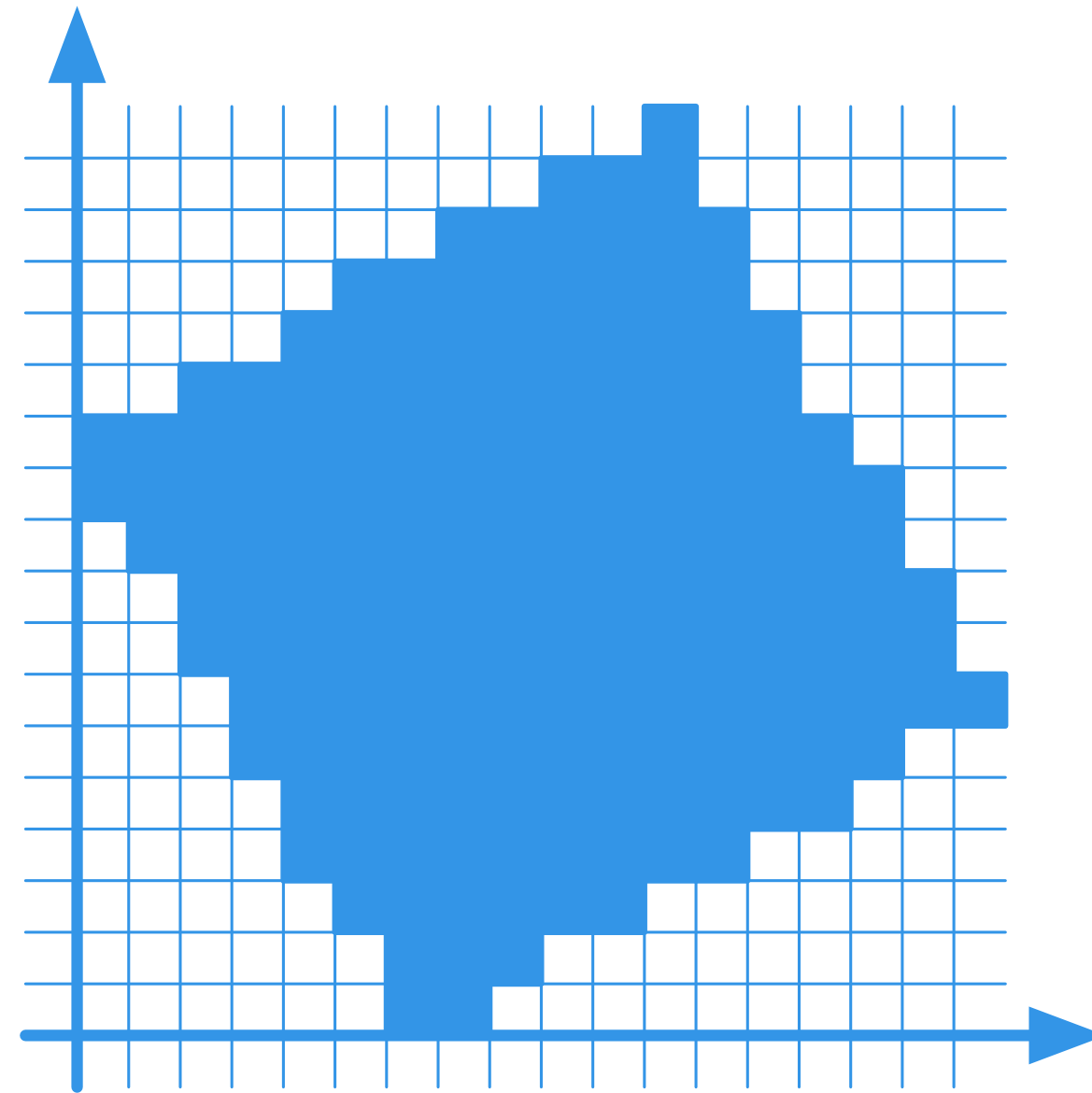
Koordinatensysteme



Koordinatensysteme

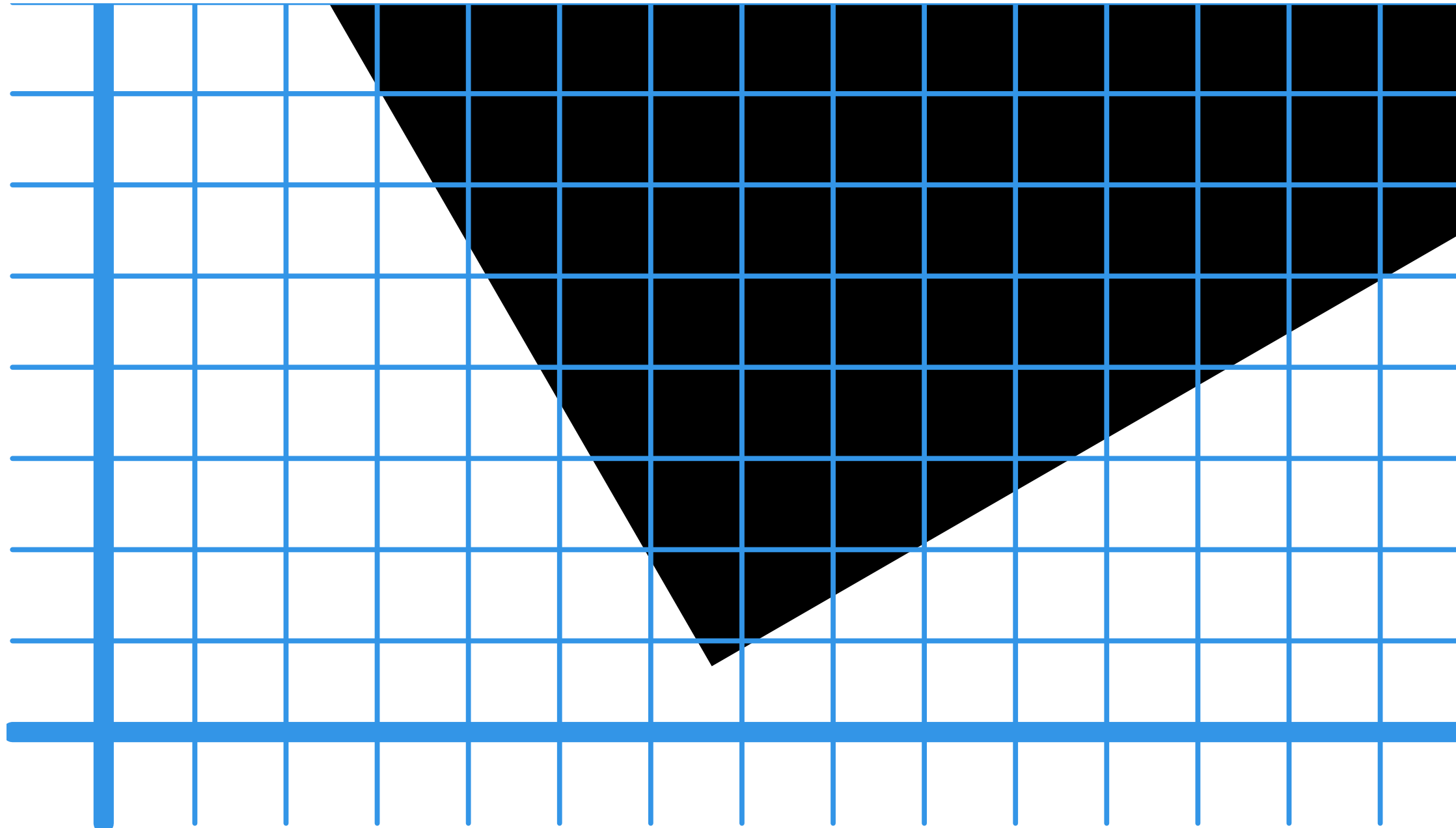


Koordinatensysteme

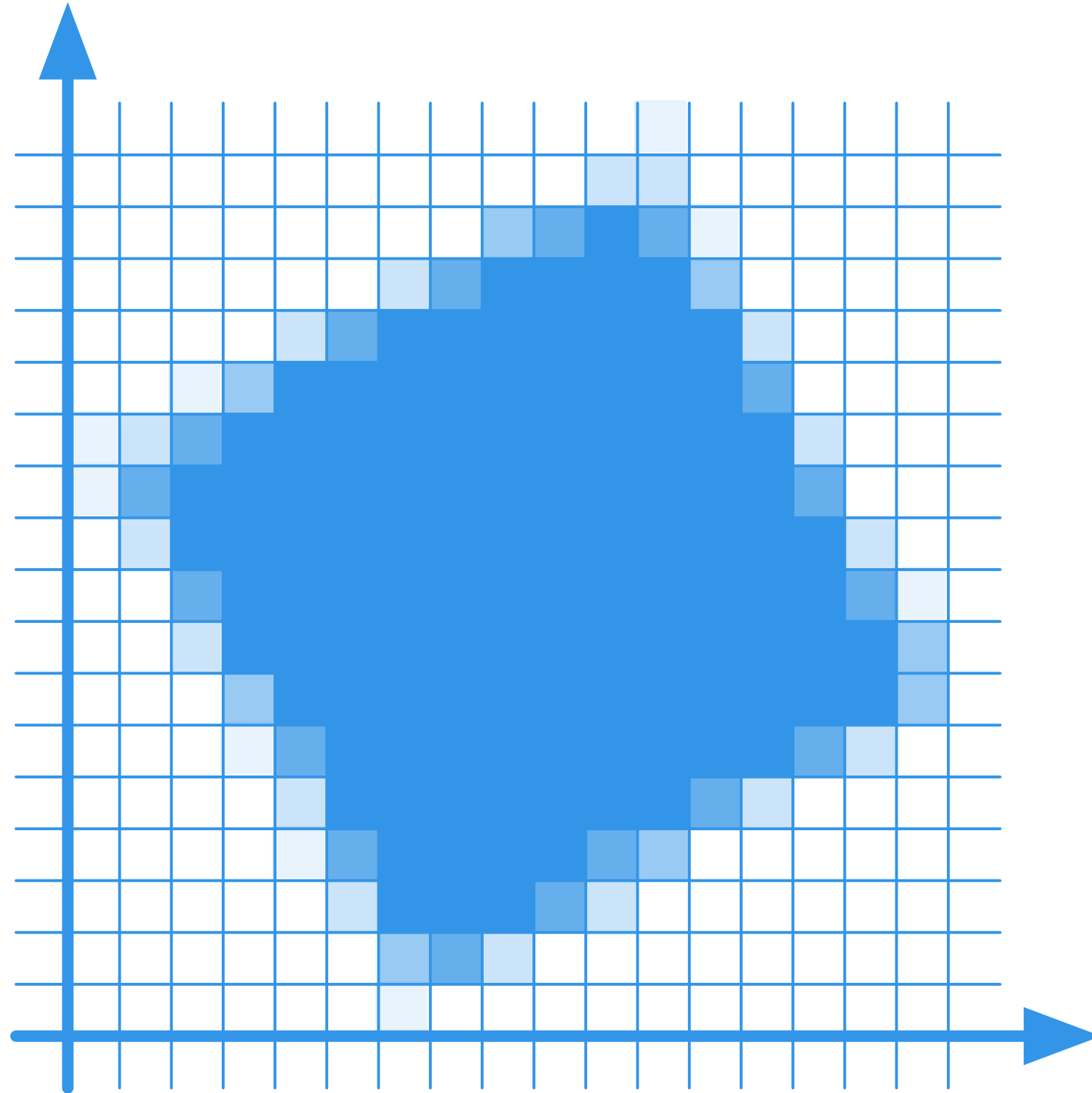


Device Space

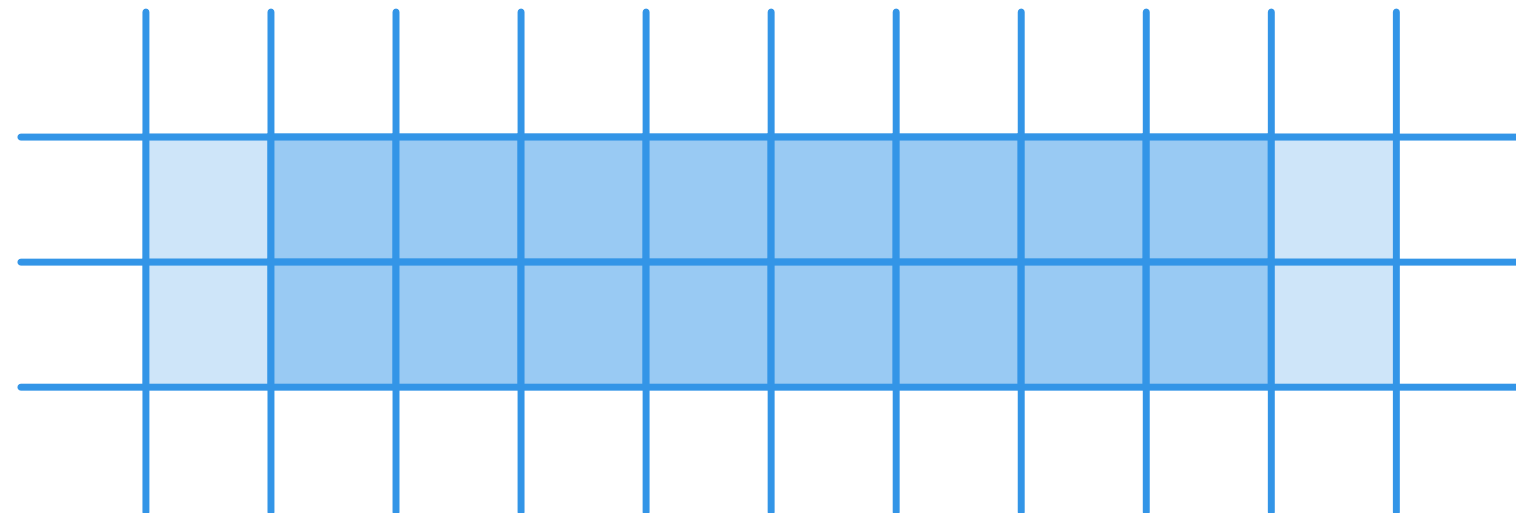
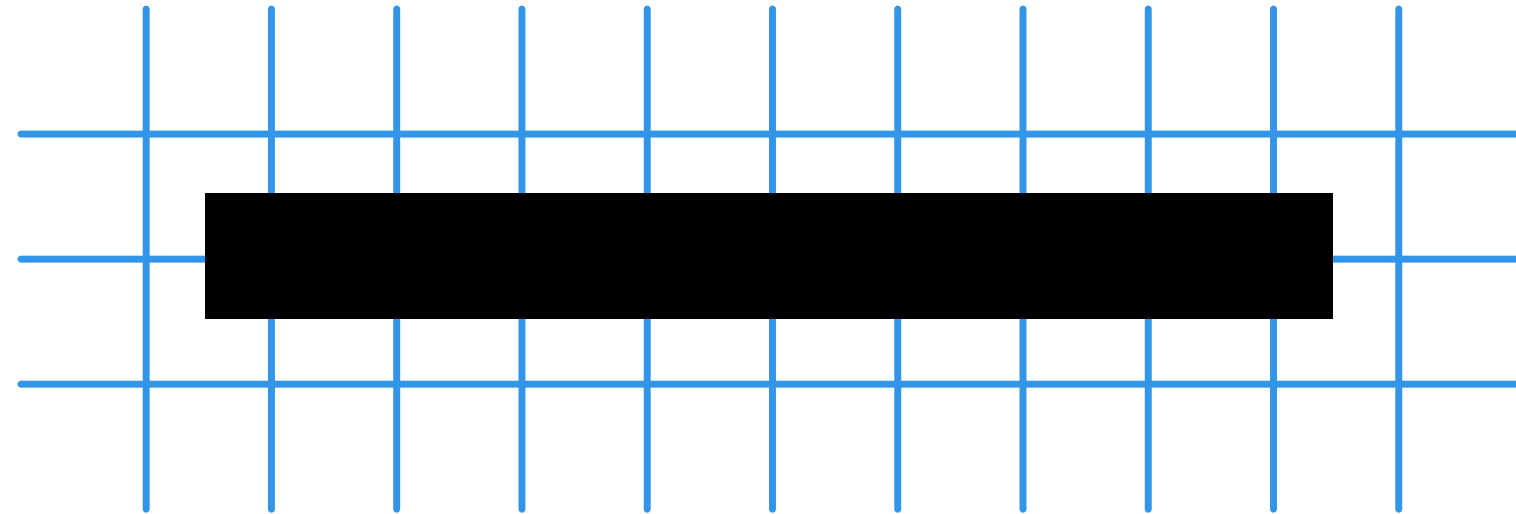
Bildschirm-Kantenglättung



Bildschirm-Kantenglättung



Bildschirm-Kantenglättung



Bildschirm-Kantenglättung

Text unter iOS

Maßnahmen

Bildschirm-Kantenglättung

Text unter Mac OS



Sub-Pixel Rendering

Auflösungsunabhängigkeit

Auflösungsunabhängigkeit

- einer einzelnen Ansicht
- einer gesamten Anwendung
- des gesamten Desktops / Bildschirms

Auflösungsunabhängigkeit

Einer einzelnen Ansicht

- Geschichte: TeX 1978
Metafont – DVI (Device Independent File Format)
- Vertaut in Quartz: z.B. Zoom innerhalb einer Ansicht
 - nutzt Skalierungs-Transformation im User Space
 - Herkömmlich: bei 100%-Zoom: 1 Point \triangleq 1 Pixel
- Drucken

Auflösungsunabhängigkeit

Einer einzelnen Ansicht

Quartz-Transformationen

```
void CGContextScaleCTM (  
    CGContextRef c,  
    CGFloat sx,  
    CGFloat sy  
);
```

```
void CGContextTranslateCTM (  
    CGContextRef c,  
    CGFloat tx,  
    CGFloat ty  
);
```

```
void CGContextRotateCTM (  
    CGContextRef c,  
    CGFloat angle  
);
```

Auflösungsunabhängigkeit

```
- (void)drawLayer:(CALayer*)layer inContext:(CGContextRef)context {  
  
    // Hintergrund weiss füllen  
    CGContextSetRGBFillColor(context, 1.0, 1.0, 1.0, 1.0);  
    CGContextFillRect(context, self.bounds);  
  
    CGContextSaveGState(context);  
  
    // Context flippen, damit PDF aufrecht erscheint  
    CGContextTranslateCTM(context, 0.0, self.bounds.size.height);  
    CGContextScaleCTM(context, 1.0, -1.0);  
  
    // Zoom-Skalierung setzen  
    CGContextScaleCTM(context, myZoomFactor, myZoomFactor);  
    CGContextDrawPDFPage(context, myPDFPage);  
  
    CGContextRestoreGState(context);  
}
```

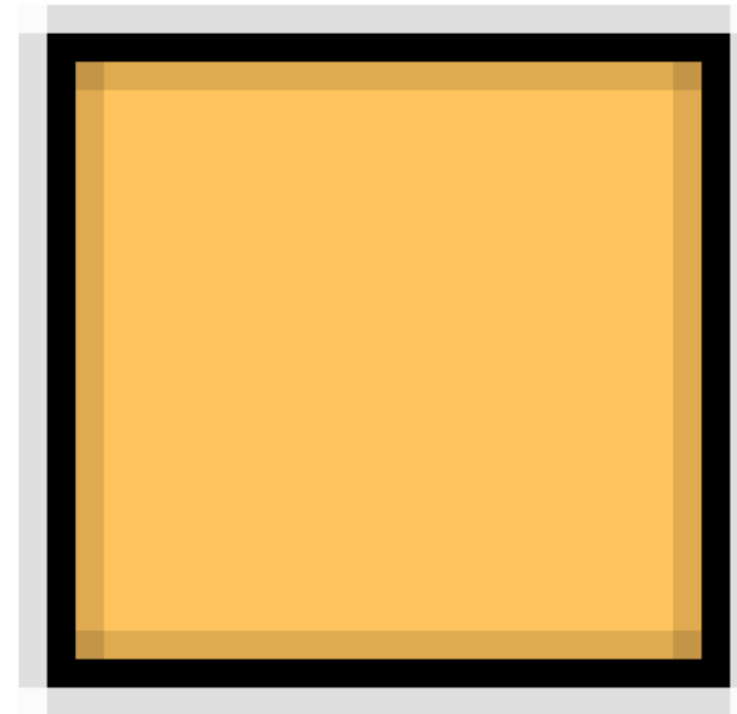
Auflösungsunabhängigkeit

Einer einzelnen Ansicht

Problem: Mit Kantenglättung werden vertikale und horizontale Linien unscharf



100% Zoom



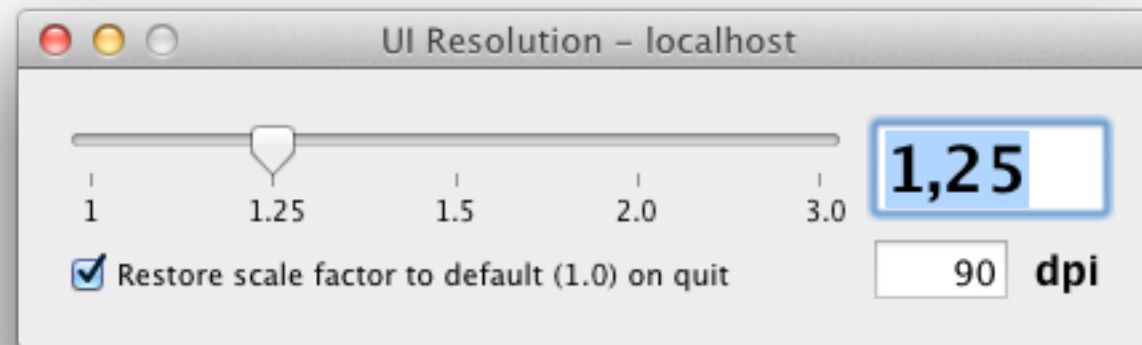
125% Zoom

Beispiel aus Apple Keynote / Omni Graffle

Auflösungsunabhängigkeit

Des gesamten Desktop

- Warum?
 - Ausgleich für Bildschirme mit hoher Punktdichte
 - Anpassen an individuelle Sehkraft oder Arbeitsweise
- Probleme
 - Senkrechte und horizontale Linien sind in der Überzahl
 - Rückwärtskompatibilität



Auflösungsunabhängigkeit

Des gesamten Desktop

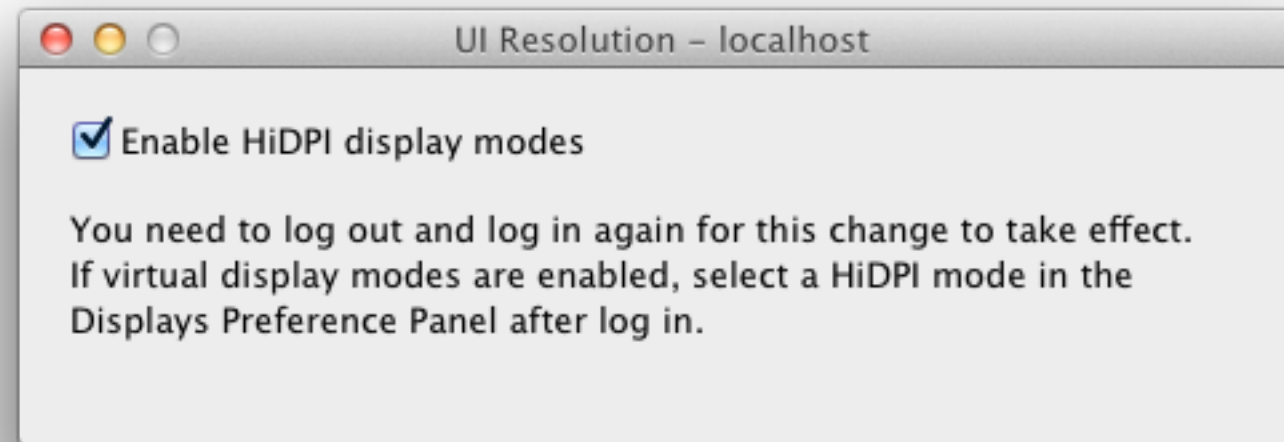
- Apples Lösung: Faktor zwei
 - 1 Point \triangleq 1 Pixel (Mac LoDPI / iPad / iPhone ohne Retina-D.)
 - 1 Point \triangleq 2 Pixel (Mac HiDPI / iPhone mit Retina-Display)
 - Achtung: Event- und Window-Koordinaten nun in Points!

Auflösungsunabhängigkeit

HiDPI-Modus in Mac OS 10.7

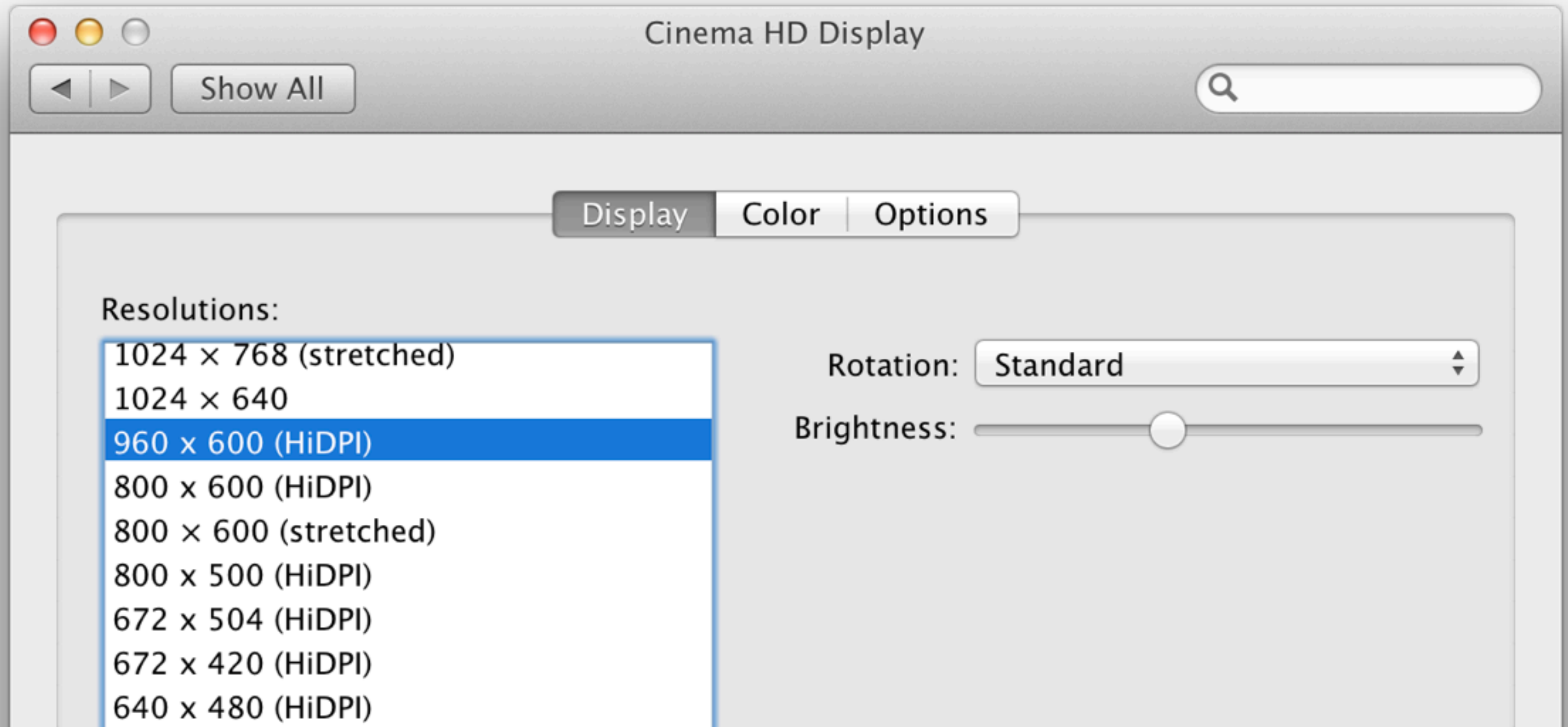


Name Quartz Debug
Kind Application
Size 3,3 MB
Created 7. April 2011 03:21
Modified 02.09.2011 11:15
Last opened 02.09.2011 11:15
Version 4.2



Auflösungsunabhängigkeit

HiDPI-Modus in Mac OS 10.7



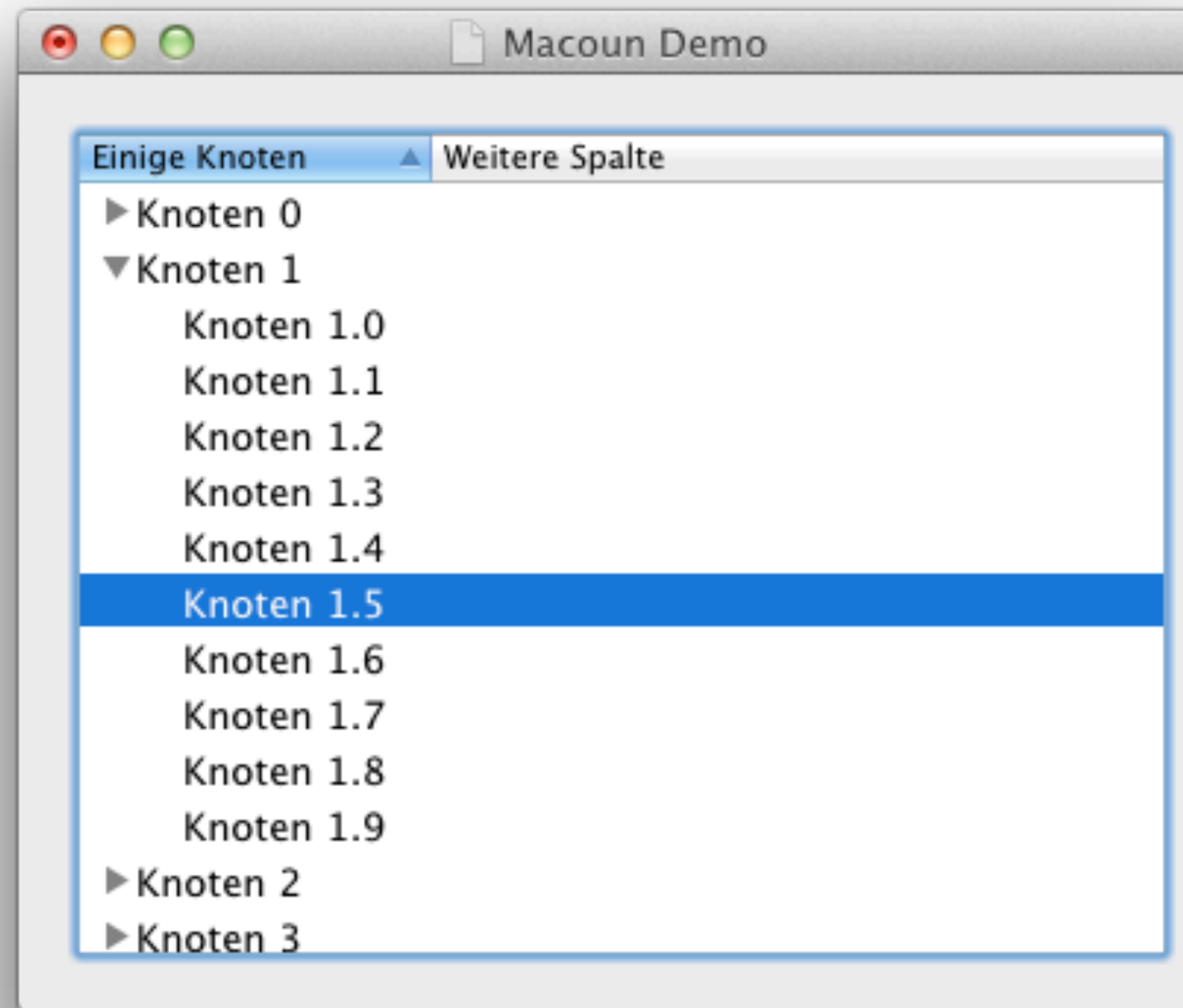
Praxis

Praxis

- Wunsch: Eine Implementierung soll alles bedienen
 - LoDPI / HiDPI Bildschirm
 - Drucken

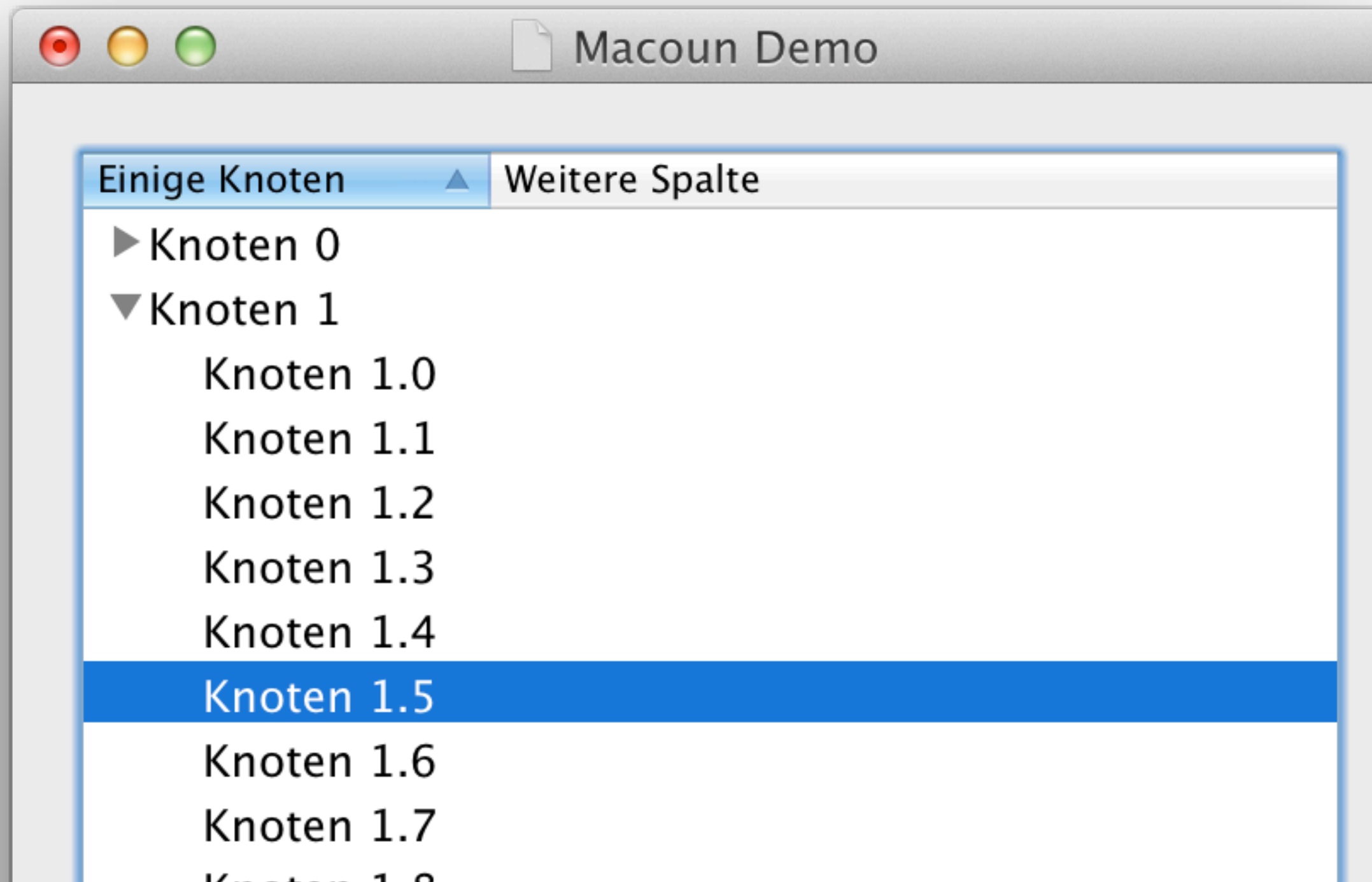
Praxis

Beispiel: NSOutlineView in LoDPI



Praxis

Beispiel: NSOutlineView in HiDPI



Praxis

Beispiel: NSOutlineView Drucken?

Demonstration

Praxis

Beispiel: NSOutlineView Drucken

- Text funktioniert einwandfrei
- Probleme
 - Liniendicken
 - Linienfarben
 - Bildgradienten
 - Rundungs-Artefakte
 - Pixelige Bilder

Praxis

Pixelige Bilder

- `-[UIImage imageNamed:@"MeinBild"]`
- Namenskonvention für Bitmap-Bilder:
 - LoDPI: `<Bildname>.<Typ-Suffix>`
 - HiDPI: `<Bildname>@2x.<Typ-Suffix>`
- Drucken
 - noch höher auflösende Bitmap-Variante
 - PDF
 - Gekapselte Kombination aus Bitmap und PDF

Praxis

Linien

- Anforderungen:
 - Keine Glättung bei vertikalen / horizontalen Linien
 - Bildschirm- und Druckerauflösung soll ausgenutzt werden
 - Linien, die dünner als Bildschirmauflösung sind, sollen durch z.B. Aufhellen simuliert werden

Praxis

Wie verhindert man Linienglättung?

- Schlechte Ideen:
 - `CGContextSetShouldAntialias(context, FALSE)`
 - `CGContextMoveToPoint(context, x + 0.5, y + 0.5)`
- Bessere Ideen?

Praxis

Wie zeichnet man eine horizontale Linie?

Versuch eine allgemeinen Snapping-Lösung zu implementieren:

```
CGPoint PWSnapPointToPixel (CGContextRef ctx,
                             CGPoint point,
                             PWSnapToPixelFormatMode hMode,
                             PWSnapToPixelFormatMode vMode) {
    PWResolveSnapModes (ctx, &hMode, &vMode);

    point = CGContextConvertPointToDeviceSpace (ctx, point);

    if (hMode == PWSnapToFullPixel)
        point.x = floor (point.x + 0.5);
    else if (hMode == PWSnapToHalfPixel)
        point.x = floor (point.x) + 0.5;

    if (vMode == PWSnapToFullPixel)
        point.y = floor (point.y + 0.5);
    else if (vMode == PWSnapToHalfPixel)
        point.y = floor (point.y) + 0.5;

    return CGContextConvertPointToUserSpace (ctx, point);
}
```

Praxis

Quartz-Konvertierungsfunktionen

```
CGPoint CGContextConvertPointToDeviceSpace (CGContextRef c, CGPoint point);
CGPoint CGContextConvertPointToUserSpace   (CGContextRef c, CGPoint point);

CGSize CGContextConvertSizeToDeviceSpace   (CGContextRef c, CGSize size);
CGSize CGContextConvertSizeToUserSpace     (CGContextRef c, CGSize size);

CGRect CGContextConvertRectToDeviceSpace   (CGContextRef c, CGRect rect);
CGRect CGContextConvertRectToUserSpace     (CGContextRef c, CGRect rect);
```

Praxis

Cocoa-Konvertierungsfunktionen

```
@interface NSView
```

- (NSPoint)convertPointToBacking:(NSPoint)point;
- (NSPoint)convertPointFromBacking:(NSPoint)point;

- (NSSize)convertSizeToBacking:(NSSize)size;
- (NSSize)convertSizeFromBacking:(NSSize)size;

- (NSRect)convertRectToBacking:(NSRect)rect;
- (NSRect)convertRectFromBacking:(NSRect)rect;

```
@end
```

```
CGPoint PWSnapPointToPixel (CGContextRef ctx,
                           CGPoint point,
                           PWSnapToPixelMode hMode,
                           PWSnapToPixelMode vMode) {
    PWResolveSnapModes (ctx, &hMode, &vMode);

    point = CGContextConvertPointToDeviceSpace (ctx, point);

    if (hMode == PWSnapToFullPixel)
        point.x = floor (point.x + 0.5);
    else if (hMode == PWSnapToHalfPixel)
        point.x = floor (point.x) + 0.5;

    if (vMode == PWSnapToFullPixel)
        point.y = floor (point.y + 0.5);
    else if (vMode == PWSnapToHalfPixel)
        point.y = floor (point.y) + 0.5;

    return CGContextConvertPointToUserSpace (ctx, point);
}
```



```

void PWResolveSnapModes(CGContextRef ctx,
                        PWSnapToPixelMode* inOutHMode, // mode or line width
                        PWSnapToPixelMode* inOutVMode) // mode or line width
{
    CGSize lineWidths = CGSizeZero;
    BOOL hasLineWidths = NO;
    if (*inOutHMode > 0.0) {
        lineWidths.width = *inOutHMode;
        hasLineWidths = YES;
    }

    // ...das gleiche für VMode

    if (hasLineWidths) {
        lineWidths = CGContextConvertSizeToDeviceSpace (ctx, lineWidths);
        if (*inOutHMode > 0.0) { // 1.0 -> half, 2.0 -> full, 3.0 -> half
            double remainder = fmod (fabs (lineWidths.width), 2.0);
            *inOutHMode = (remainder == 0.0 || remainder > 1.0) ?
                PWSnapToFullPixel : PWSnapToHalfPixel;
            // ...das gleiche für VMode
        }
    }
}

```

Praxis

Zeichnen einer horizontalen Linie

```
CGPoint startPoint = CGPointMake(xStart, y);
CGPoint endPoint   = CGPointMake(xEnd,   y);

startPoint = PWSnapPointToPixel(ctx, startPoint, PWSnapToFullPixel, lineWidth);
endPoint   = PWSnapPointToPixel(ctx, endPoint,   PWSnapToFullPixel, lineWidth);

CGContextMoveToPoint    (ctx, startPoint.x, startPoint.y);
CGContextAddLineToPoint(ctx, endPoint.x,   endPoint.y);

// Ähnlich implementiert: PWSnapSizeToPixel, PWSnapRectToPixel
```

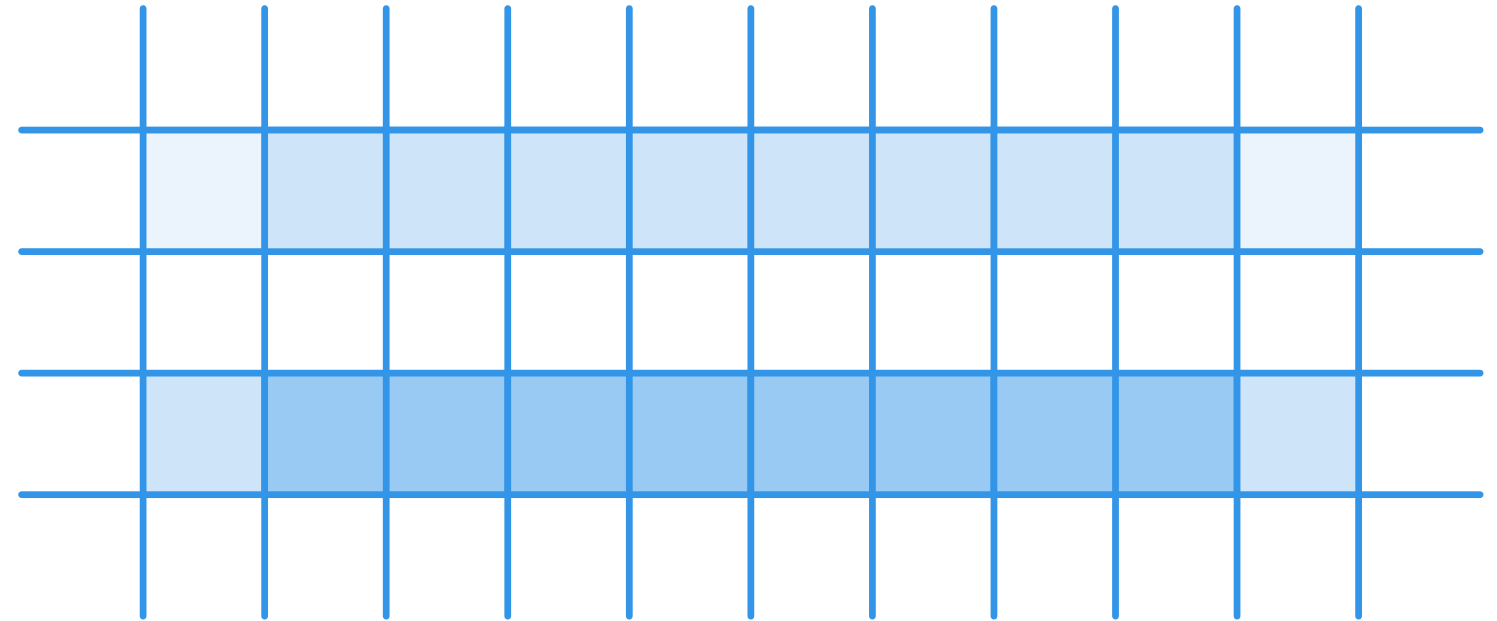
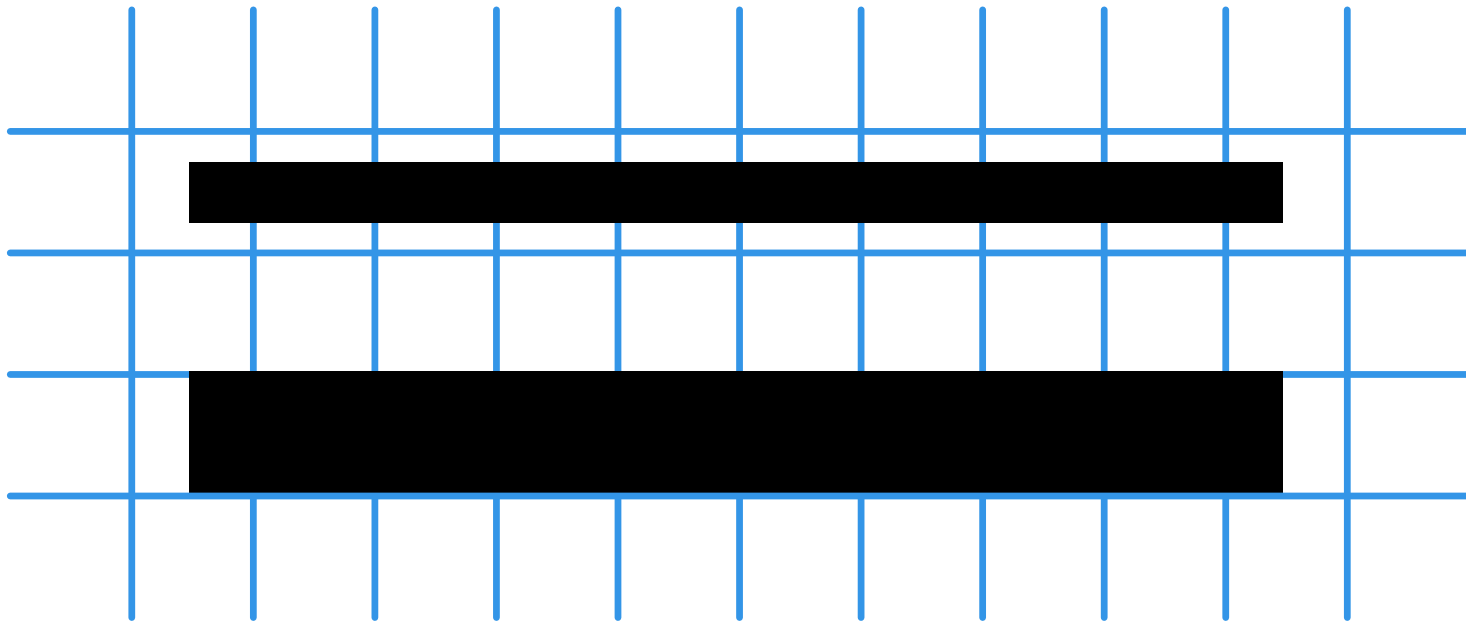
Ganz einfach...

...sind wir schon fertig?

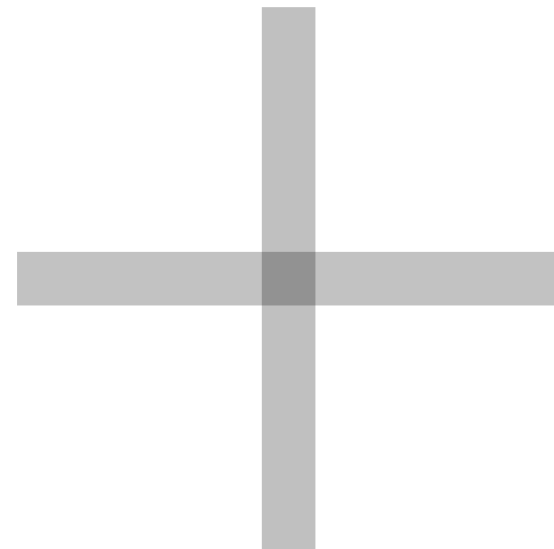
Praxis

Dünne Linien

- Quartz-Standard: Transparenz mittels Alpha-Kanal



- Führt zu Artefakten bei sich kreuzenden Linien



```
@implementation PWGraphicsContext
```

```
...
```

```
- (void) screenifyLineColor:(CGColorRef*)inOutColor  
    forWidth:(CGFloat*)inOutWidth  
    snapMode:(PWSnapToPixelMode*)outSnapMode {
```

```
    if (self.isDrawingToScreen) {  
        CGFloat pixelPerPoint          = self.pixelPerPoint;  
        CGFloat userLineWidth           = *inOutWidth;  
        CGFloat idealDeviceLineWidth = userLineWidth * pixelPerPoint;  
        BOOL isSubDeviceWidth           = (idealDeviceLineWidth < 1.0);  
        CGFloat realDeviceLineWidth     = isSubDeviceWidth ?  
            ceil(idealDeviceLineWidth) : round(idealDeviceLineWidth);  
        *inOutWidth = realDeviceLineWidth / pixelPerPoint;  
  
        if (isSubDeviceWidth && inOutColor) {  
            CGFloat lighteningFactor = idealDeviceLineWidth / realDeviceLineWidth;  
            // Kein Abblenden, wenn Effekt vernachlässigbar ist  
            if (lighteningFactor < 0.95)  
                *inOutColor = PWColorByBlendingColorToColor (self.backgroundColor,  
                                                                *inOutColor, lighteningFactor);  
        }  
    }
```

```
...
```

```
...  
  
    if (outSnapMode) {  
        double remainder = fmod (realDeviceLineWidth, 2.0);  
        *outSnapMode = (remainder == 0.0 || remainder > 1.0) ?  
                        PWSnapToFullPixel : PWSnapToHalfPixel;  
    }  
} else if (outSnapMode)  
    *outSnapMode = PWSnapToPixelOff;  
}  
...
```


...

```
- (CGFloat) pixelPerPoint
{
    CGAffineTransform t = CGContextGetCTM (cgContext_);
    if (t.a != lastT_.a || t.b != lastT_.b || t.c != lastT_.c || t.d != lastT_.d) {
        lastT_ = t;
        CGSize devSizeX = CGContextConvertSizeToDeviceSpace(ctx_, CGSizeMake (1.0, 0.0));
        CGSize devSizeY = CGContextConvertSizeToDeviceSpace(ctx_, CGSizeMake (0.0, 1.0));
        pixelPerPoint_ = (PWSizeLength (devSizeX) + PWSizeLength (devSizeY)) / 2.0;
    }
    return pixelPerPoint_;
}

- (CGPoint) snapPoint:(CGPoint)point toPixelWithMode:(PWSnapToPixelMode)mode {
    return self.isDrawingToScreen ?
        PWSnapPointToPixel (ctx_, point, mode, mode) : point;
}
```

@end

Praxis

Zeichnen einer horizontalen Linie

```
PWGraphicsContext* context = [[PWGraphicsContext alloc] initWithCGContext:ctx
                                                                    isDrawingToScreen:YES];

CGPoint startPoint = CGPointMake(xStart, y);
CGPoint endPoint   = CGPointMake(xEnd,   y);
CGColor lineColor = ...;
CGFloat lineWidth = ...;
PWSnapToPixelMode snapMode;
[context screenifyLineColor:&lineColor forWidth:&lineWidth snapMode:&snapMode];

startPoint = [context snapPoint:startPoint toPixelWithMode:snapMode];
endPoint    = [context snapPoint:endPoint   toPixelWithMode:snapMode];

CGContextMoveToPoint(context.CGContext, startPoint.x, startPoint.y);
CGContextAddLineToPoint(context.CGContext, endPoint.x,   endPoint.y);
```

Praxis

Zeichnen einer horizontalen Linie

- Animation
 - Animation ohne Kantenglättung kann zu Jitter-Artefakten führen
 - Text in `CGLayer` / `CALayer` erlaubt keine Sub-Pixel-Glättung!

Demonstration

files.me.com/illenberger/rhwvun

Fragen?

Vielen Dank