

OpenGL auf Mac OS X und iOS


Torsten Kammer

28.4.2011

Grundlagen

- API für hardwarebeschleunigte 3D-Grafik
- Kann auch für schnelles 2D verwendet werden
- Grundlage von Core Image, Core Animation, Core Video, ...

Versionen

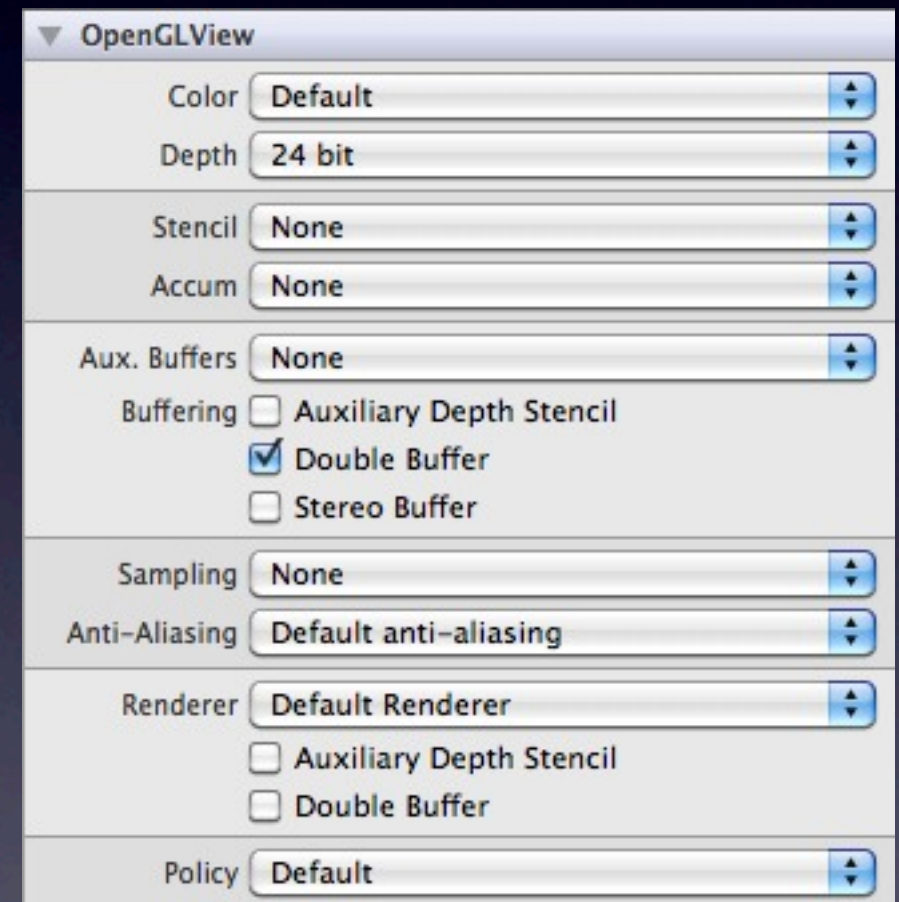
- Desktop: Verschiedene Versionen, alle abwärtskompatibel. Neueste: 4.1
- Intel-Mac: 2.1 oder höher
- iOS: OpenGL ES, Teilmenge. Immer vorhanden: OpenGL ES 1.1 
- iPhone 3GS und höher: zusätzlich OpenGL ES 2.0 - nicht abwärtskompatibel

OpenGL Kontext

- Farb-, Tiefen- und weitere Buffer (Zeichenfläche)
- Aktueller Zustand
- Ressourcen (Texturen, Geometrie) - Können von mehreren Kontexten geteilt werden.
- Mac: NSOpenGLView erzeugt automatisch einen
- iOS: EAGLContext muss manuell erstellt werden - Buffer auch.

Mac: NSOpenGLView

- Basisklasse, vereinfacht Verwaltung
- Erstellt mit Interface Builder
- Übliche Einstellungen: Double Buffer, Depth, sonst Standard
- Zeichnen: drawRect:, am Ende [[self context] flushDrawable]
- Spezialmethoden prepareGL, reshape



Grundeinstellungen

```
// Größe der View
glViewport(0, 0, width, height);

// Koordinatensystem einstellen
glMatrixMode(GL_PROJECTION);
// Alte Einstellungen löschen
glLoadIdentity();
// Links, Rechts, Unten, Oben, Nah, Fern
glOrtho(0, width, 0, height, -1, 1);

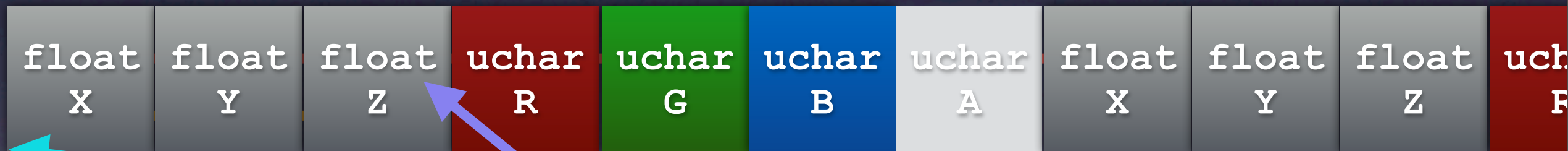
// Modus zurücksetzen
glMatrixMode(GL_MODELVIEW);
```

Zeichnen

- Nur Punkte, Linien, Dreiecke - aber davon viele auf einmal
- Farbe pro Eckpunkt (Verläufe automatisch)
- Viele Datentypen möglich für Positionen, Farben usw.

Zeichnen: Vertex Arrays

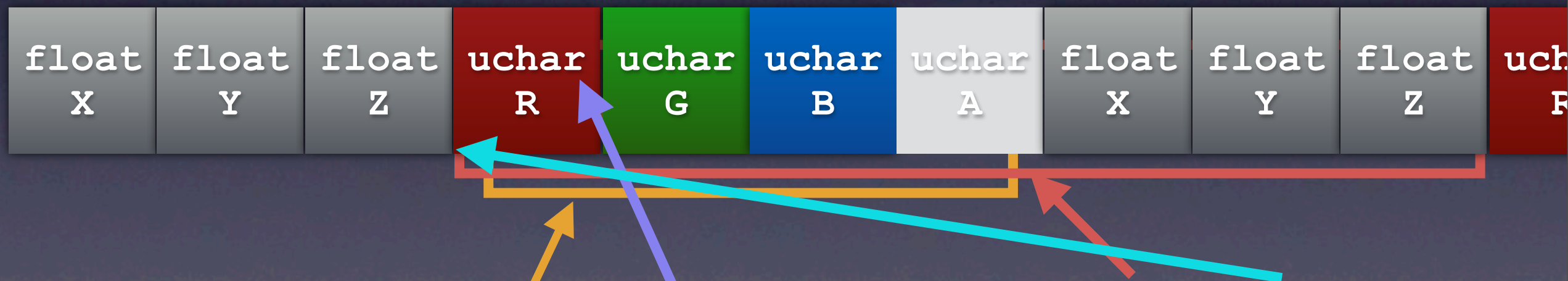
- Beste (iOS: Einzige) Art zu zeichnen
- Positionen, Farben etc: Ein oder mehrere Arrays. Getrennt oder interleaved. Apple empfiehlt interleaved



```
glVertexPointer(3, GL_FLOAT, 16, p);
```


Zeichnen: Vertex Arrays

- Beste (iOS: Einzige) Art zu zeichnen
- Positionen, Farben etc: Ein oder mehrere Arrays. Getrennt oder interleaved. Apple empfiehlt interleaved



```
glColorPointer(4, GL_UNSIGNED_CHAR, 16, p+12);
```

Beispiel: Ein Dreieck

```
// Vertex-Arrays anschalten
```

```
glEnableClientState(GL_VERTEX_POINTER);
```

```
glEnableClientState(GL_COLOR_POINTER);
```

```
// Daten          Position  Farbe (RGBA)
```

```
GLfloat data[] = {0, 0,      1, 0, 0, 1,  
                  0, 100,    0, 1, 0, 1,  
                  100, 0,    0, 0, 1, 1};
```

```
// Pointer
```

```
glVertexPointer(3, GL_FLOAT, 24, data);
```

```
glColorPointer(4, GL_FLOAT, 24, &(data[2]));
```

```
// Zeichnen
```

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```


Demo

iOS

- Subklasse von normaler UIView
- Layer Class: CAEAGLLayer
- Kontext: EAGLContext selbst erzeugen
- Framebuffer muss selbst erstellt werden

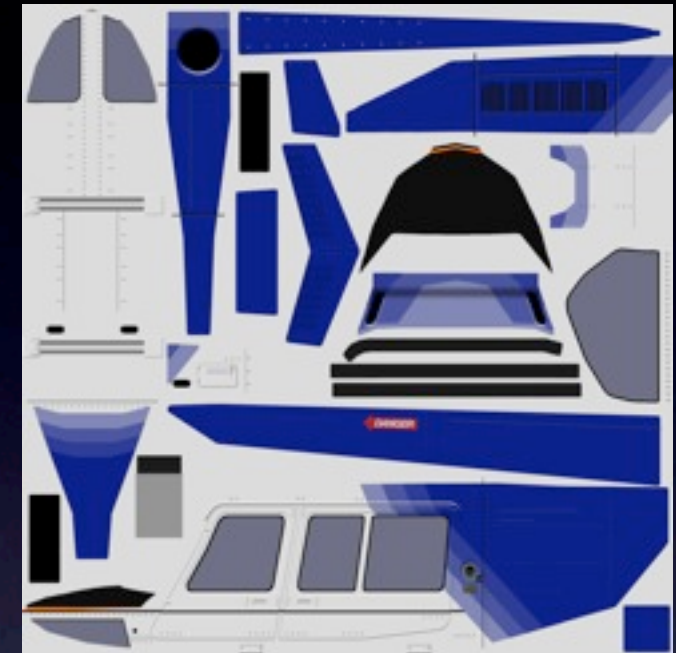
Demo

Transformationen

- Analog zu CGContext...CTM
- `glLoadIdentity()` - Zurücksetzen
- `glTranslatef(x, y, z)` - Verschieben
- `glRotatef(grad, achseX, achseY, achseZ)` - um Nullpunkt drehen
- `glScalef(x, y, z)` - skalieren

Texturen

- Mehr Details: Bilder auf Dreiecke bringen
- Anordnen auf Polygonen mit Texturkoordinaten
- Zwei Modi (müssen getrennt an- und abgeschaltet werden):
 - GL_TEXTURE_2D: Seitenlänge Zweierpotenz, Koordinaten 0...1
 - GL_TEXTURE_RECTANGLE_ARB/_OES: Seitenlängen frei, Koordinaten in Pixeln



```
glEnable(GL_TEXTURE_2D);

GLuint textureID;
glGenTextures(1, &textureID);

glBindTexture(GL_TEXTURE_2D, textureID);

// Verhalten bei vergrößern/verkleinern
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Verhalten an den Rändern
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

// Daten hochladen
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, size, size, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bytes);

// Anwenden
glBindTexture(GL_TEXTURE_2D, textureID);
glDrawArrays(...)
```


Mehr Performance: VBOs

- Daten der Vertexarrays in Grafikspeicher hochladen
- Erzeugen: glGenBuffers
- Aktuellen VBO setzen: glBindBuffer (GL_ARRAY_BUFFER, id)
- Daten hochladen: glBufferData(GL_ARRAY_BUFFER, length, data, GL_STATIC_DRAW);
- Anwenden: Wie normale Vertex Arrays, aber Pointer ist jetzt Offset in zuletzt gebundenen VBO

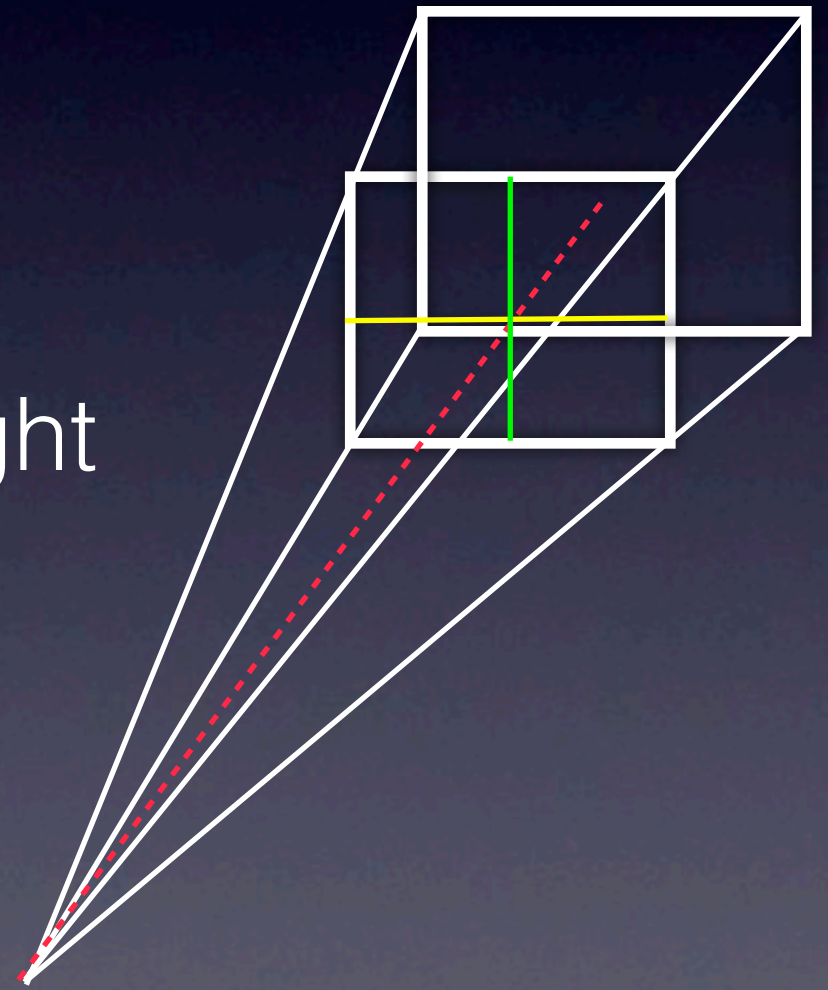
Demo

3D-Grafik

- Klar: Drei Komponenten für Vertex Array
usw...
- Depth Test: Speichere Entfernung Pixel-Kamera. Erlaube nur neue Pixel die näher sind. vermindert Überlappung
- Kamera: Frustrum-Projektion

Frustum Projektion

- Für perspektivische Darstellung
- `glFrustum(left, right, bottom, top, near, far)`
- Normalfall: $\text{bottom} = -\text{top}$, $\text{left} = -\text{right}$
- $|\text{top}|/|\text{left}| = \text{Aspektrate}$



Demo

Mac OS X Tricks

- Debugging
- Texturen
- Kontext
- Full Screen

OpenGL Profiler

- Debuggen von OpenGL
- Breakpoints
- Ressourcen und Buffer
- Leider nicht für iPhone
- Alternative: gDEDebugger - auch kein iPhone

Texturen Laden

- Am bequemsten: CoreGraphics und ImageIO
- Workflow:
 - CGImageSource -> Breite, Höhe, CGImage
 - Byte Array mit $(4 \cdot \text{Breite} \cdot \text{Höhe})$ Einträgen
 - CGBitmapContext erstellen (dafür Color Space)
 - Bild im Kontext zeichnen
 - Alles bis auf Buffer löschen


```

CFURLRef url;

CGImageSourceRef source = CGImageSourceCreateWithURL((CFURLRef) fileURL, NULL);
CFDictionaryRef dict = CGImageSourceCopyPropertiesAtIndex(source, 0, NULL);
CFIndex width, height;
CFNumberGetValue(CFDictionaryGetValue(dict, kCGImagePropertyPixelWidth), kCFNumberCFIndexType,
&width);
CFNumberGetValue(CFDictionaryGetValue(dict, kCGImagePropertyPixelHeight), kCFNumberCFIndexType,
&height);
CFRelease(dict);

CGImageRef image = CGImageSourceCreateImageAtIndex(source, 0, NULL);
CFRelease(source);

unsigned char *data = malloc(width * height * 4);

CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
CGContextRef context = CGContextCreate(data, width, height, 8, width * 4, colorSpace,
kCGImageAlphaPremultipliedLast);
CGColorSpaceRelease(colorSpace);

CGContextDrawImage(context, CGRectMake(0.0f, 0.0f, (CGFloat) width, (CGFloat) height), image);
CGContextRelease(context);
CGImageRelease(image);

GLuint textureID;
glGenTextures(1, &textureID);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, size, size, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);

free(data);

```

Kontextverwaltung

- Viele Views verwenden OpenGL intern (z.B. UIImageView)
- Kontext ist statische globale Variable - andere Views setzen ihren eigenen
- Einfach: `[[self openGLContext] makeCurrent]`
- Etwas Eleganter: Makros

Makros für Kontext

- Variable `cgl_ctx` muss vorhanden sein (z.B. als Instanzvariable)
- `cgl_ctx = [[self openGLContext] CGLContextObj]`
- `#include <OpenGL/CGLMacro.h>`

Core Image

- Verwendet intern OpenGL. Lässt sich sehr gut integrieren.
- Am sinnvollsten zusammen mit Frame Buffer Objects - erlaubt CImage als Textur zu verwenden

Prinzipiell

- CIColorContext mit `-contextWithCGLContext: pixelFormat:colorSpace:options:` erstellen
- Leere Textur erzeugen, FBO erstellen und mit Textur verbinden
- Rendern in FBO aktivieren, GL-Status für Core Image setzen
- CIColorContext `-drawImage:inRect:fromRect`
- Textur enthält Ergebnis