Texturen in Open GL

- Als erstes muss eine Textur auf die Graphikkarte geladen werden:

```
glTexImage{1,2}D( target, level, internal, width,
                  [height,] border, format, type, data )
```

`target` = `GL_TEXTURE_1D`, `GL_TEXTURE_2D`, ...

`level` = 0 bzw. der zu definierende MipMap Level (später)

`internal` = Anzahl der Komponenten der Textur: 1, 2, 3, 4, `GL_RGB`, `GL_LUMINANCE`, `GL_R3_G3_B2`...

`width` / `height` = Breite / Höhe, **muß** = 2^n+2*border sein

`(gluScaleImage()` kann Bilder skalieren helfen)

`border` = Breite des Randes, 0 oder 1

`format` = was steht pro Pixel im Speicher: `GL_RGB`, `GL_RGBA`, ...

`type` = Typ der Pixel: `GL_UNSIGNED_BYTE`, `GL_FLOAT`, ...

`data` = Adresse der Pixeldaten im Hauptspeicher

- Textur einschalten:

```
glEnable( GL_TEXTURE_{12}D )
```

- Zu jedem Eckpunkt gehört eine Texturkoordinate:

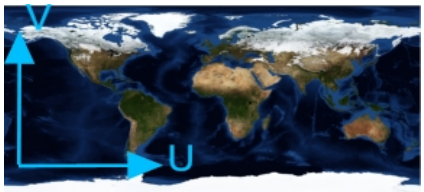
```
glTexCoord{1234}f[v]( value )
```

  - das Bild liegt dabei im Bereich [0,1]x[0,1]
  - im Normalfall werden nur die ersten beiden (u und v) verwendet
    - die dritte (q) wird für 3D-Texturen benötigt, die vierte (r = wie die homogene Koordinaten) nur für Spezialeffekte

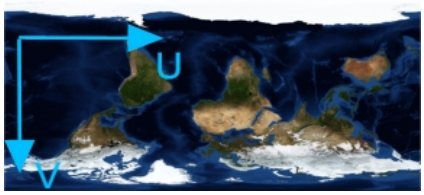
- Achtung: OpenGL hat keinen Image-Loader!
  - Aber: Qt bietet hier Funktionen an (oder andere Libs)
  - Oder: `glCopyTexImage2D(...)` liest Bild aus Framebuffer in Texturspeicher

Orientierung

- Der Fluch der Orientierung:
  - OpenGL Orientierung



  - Orientierung des Bild-Arrays nach dem Laden



- Achtung: Qt's **`bindTexture`** spiegelt das Bild, bevor es zur Graphikarte geschickt wird! Evtl. besser "von Hand" binden ...

Die Texturmatrix

- Neben den Matrizen **`GL_MODELVIEW`** und **`GL_PROJECTION`** unterstützt OpenGL eine eigene „globale“ Matrix für Texturen:

```
glMatrixMode( GL_TEXTURE )
```

- Die Texturkoordinaten werden vor Benutzung mit dieser Matrix multipliziert
- Anwendung: sich bewegende Texturen, z.B. Wellen auf einer Oberfläche

Beeinflussung der Pixelfarbe in OpenGL

- Funktion:

```
glTexEnvi( GL_TEXTURE_ENV,
           GL_TEXTURE_ENV_MODE, value )
```

- 4 Möglichkeiten für ***value***:
  - **GL_REPLACE**: Texelfarbe ersetzt Pixelfarbe (am häufigsten)
  - **GL_MODULATE**: komponentenweise Mult. von *T* und *F*

$$T_{RGB} \cdot F_{RGB}$$

  - **GL_DECAL**:

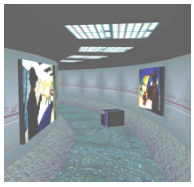
$$\alpha_T \cdot T_{RGB\alpha} + (1 - \alpha_T) \cdot F_{RGB\alpha}$$

  - **GL_BLEND**:

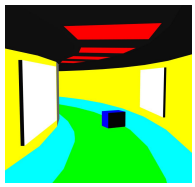
$$F_{RGB} \cdot (1 - T_{RGB}) + C_{RGB} \cdot T_{RGB}$$

  und *C* wird definiert über

```
glTexEnvfv( GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_COLOR, value )
```



T = Texelfarbe



F = Pixelfarbe ohne Textur

Koordinaten-Wrap

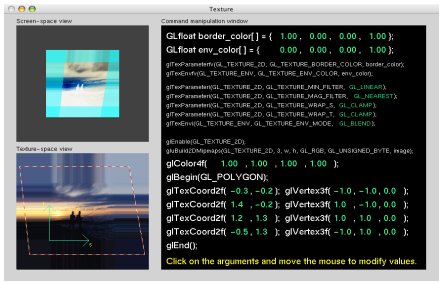
- Was geschieht, wenn Texturkoordinaten außerhalb [0,1]x[0,1] definiert werden?

```
glTexParameteri( GL_TEXTURE_{12}D, name, value )
```

  name = GL_TEXTURE_WRAP_{ST}

  value = GL_CLAMP: Werte <0 werden auf 0, Werte >1 auf 1 gezogen

  value = GL_REPEAT: nur der Nachkommaanteil wird verwendet



http://www.xmission.com/~nate/tutors.html

Textur-IDs

- Während des Renderings einer Szene benötigt man viele verschiedene Texturen
- Jedesmal **`glTexImage2D()`** ist ineffizient
- Lösung: alle Texturen gleichzeitig auf der Karte halten
- IDs generieren:

  `glGenTextures( GLint n, GLuint * indices )`

  findet **`n`** unbenutzte Textur-IDs und legt sie in **`indices`** ab
- Umschalten der aktuell aktiven Textur:

  `glBindTexture( GL_TEXTURE_{12}D, GLuint id )`
- Achtung: **dadurch werden alle Textur-relevanten Teile des Zustandes umgeschaltet!**

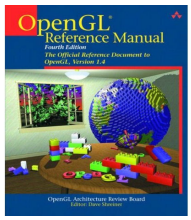
- Zusammen:

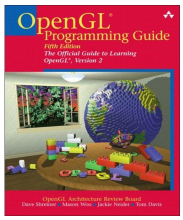
```
unsigned int tex[N];
glGenTextures( N, tex );
glBindTexture( GL_TEXTURE_2D, tex[0] );
pixels = loadImage(…);
glTexImage2D( GL_TEXTURE2D,
              0,            // mipmap level
              3,            // components [1,2,3,4]
              width, height, border,
              format,       // of the pixel data (GL_RGB..)
              type,         // GL_FLOAT...
              pixels );     // the data
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
...                         // more params (e.g. glTexEnv)
glBindTexture( GL_TEXTURE_2D, tex[1] );
pixels = loadImage(...);
glTexImage2D( GL_TEXTURE2D, ...);
```

```
// 1-tes Objekt
glBindTexture( GL_TEXTURE_2D, tex[0] );
glBegin( GL_... )
   glTexCoord2f(...);
   glNormal3f(...);
   glVertex3f(...);
   ...
glEnd();
// 2-tes Objekt
glBindTexture( GL_TEXTURE_2D, tex[1] );
glBegin( GL_... )
   glTexCoord2f(...);
   glNormal3f(...);
   glVertex3f(...);
   ...
glEnd();
```


Zum Nachlesen

- Texturierung an sich ist eine sehr mächtige (und etwas komplexe) Technik
- Texturierung in OpenGL ist — zwangsläufig — etwas komplexer als die meisten anderen Teile des APIs
- Besser vor einer Implementierung nochmals nachlesen





Auch als HTML auf der Homepage der CG-1-Vorlesung

Man Pages

Oder im Netz unter http://www.opengl.org/sdk/docs/man/