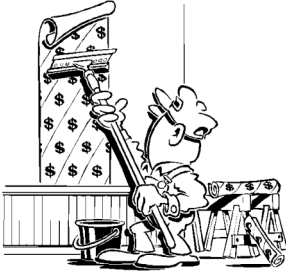


Computer-Graphik II

Texturen




G. Zachmann
Clausthal University, Germany
cg.in.tu-clausthal.de



Motivation

- Was fehlt? ...



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 2


■ ... Oberflächendetails



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 3

Grundidee

■ Objekt mit Textur „tapezieren“
 → Visuelles Detail trotz grober Geometrie



Objekt (Geometrie) Textur (Farbe)

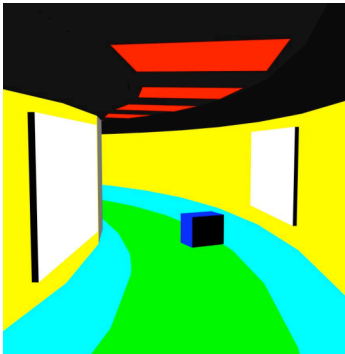
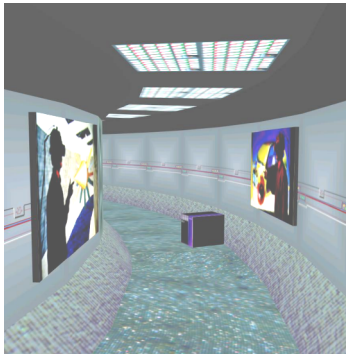
■ Ursprung: Catmull (1974), Blinn and Newell (1976), u.a.

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 4

- Großes Spektrum geometrischer Formen und physikalischer Materialien:
 - Maserungen und Muster (Holz, Marmorplatten und Tapeten)
 - Wolken
 - Strukturen unebener Oberflächen (Putzwände, Leder, Schale/Rinde von Orangen, Baumstämme, etc.)
 - Objekte im Hintergrund (Häuser, Maschinen, Pflanzen und Personen)
- Solche Objekte durch Flächen nachzubilden ist in der Regel viel zu aufwendig
- Mit Texturen kann man Objekte **visuell komplexer** gestalten:
 - Die Wand kann durch ein Rechteck modelliert werden und die Tapete wird **als Bild** aufgebracht
- Dies nennt man **Texturierung**

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 5

Weitere Beispiele

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 6

- Kaustik durch Texturen verstärkt den Unterwassereindruck



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 7


Übersicht

- Arten von Texturen: **diskret** oder **prozedural**
- **Dimension** der Texturen: 1D, 2D, 3D, 4D(?)
- Wichtige Punkte bei den diskreten 2D-Texturen:
 - **Interpolation** der Texturkoordinaten
 - **Anwendung** der Textur auf die **Beleuchtung** o. a. Oberflächeneigensch.
 - **Parametrisierung** der Fläche
 - **Filterung**
- Wie funktioniert es in OpenGL
- **Environment-Mapping**


G. Zachmann Computer-Graphik 2 - SS 07 Texturen 8

Texturarten

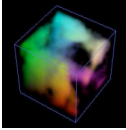
1D Texturen




2D Texturen



3D Texturen



Cubemap
Texturen



- Textur kann als Funktion einer, zwei oder drei Variablen oder als Funktion einer Richtung gesehen werden


G. Zachmann Computer-Graphik 2 - SS 07
Texturen 9


Einfacher Fall: 3D-Texturen

- 3D-Texturen nennt man auch Festkörper-Texturen (z.B. Holz und Marmor) („*solid texture*“)
- Die Textur ist an **jedem** Punkt im Raum definiert
- Die **lokalen Koordinaten** der Obj.oberfläche (x,y,z) indizieren direkt die Textur:

$$(r, g, b) = C_{\text{tex}}(x, y, z)$$
- Das Objekt wird quasi aus dem Texturvolumen "**herausgeschnitzt**"

2D-
Texturierung






3D-
Texturierung

G. Zachmann Computer-Graphik 2 - SS 07
Texturen 10

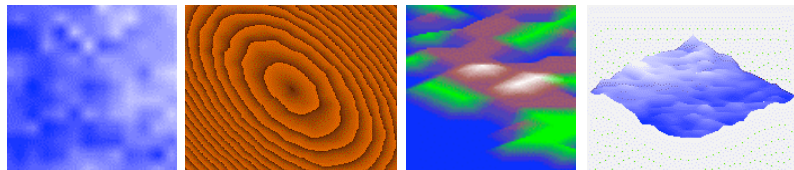
Beispiele:



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 11

Diskrete und prozedurale Texturen

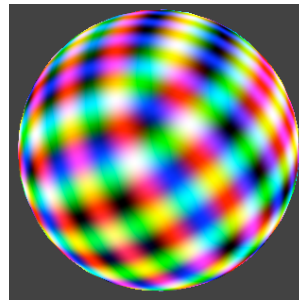
- Man unterscheidet diskrete und prozedurale Texturen
- Eine **diskrete 3D-Textur** = 3-dimensionales Array $C[i,j,k]$
 - $C[i,j,k]$ = Vektor mit 3 Farbkomponenten, ein „**Texel**“ (*texture element*)
 - Pro Pixel verwendet man (x,y,z) zum Indizieren in das Array
- Prozedurale Texturen** werden bei jedem Auslesen aus math. Funktion oder fraktalem Algorithmus berechnet

$$C_{\text{tex}}(x, y, z) := f(x, y, z)$$


G. Zachmann Computer-Graphik 2 - SS 07 Texturen 12

- Einfaches Beispiel für eine prozedurale 3D-Textur:

$$C = \begin{pmatrix} \frac{1}{2}(1 + \sin(\frac{\pi}{w_x}P_x)) \\ \frac{1}{2}(1 + \sin(\frac{\pi}{w_y}P_y)) \\ \frac{1}{2}(1 + \sin(\frac{\pi}{w_z}P_z)) \end{pmatrix}$$



- **Vorteile** der prozeduralen Texturen:
 - Speicheraufwand ist minimal
 - Texturwerte können an jeder Stelle (u,v) , bzw. (u,v,w) berechnet werden
 - Optimale Genauigkeit (kein Runden von Koord., keine Interpolation)
 - Texturen sind im gesamten Raum definiert (kein Wrap-Around / Clamping)
- **Nachteile:**
 - Schwer zu erzeugen (selbst für Experten)
 - Mindestens Grundkenntnisse der Fourier-Synthese, bzw. fraktaler Geometrie erforderlich
 - Komplexere Texturen nahezu unmöglich
 - Kosten rel. viel Zeit (Echtzeit?)

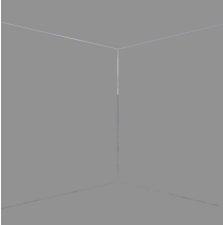
Diskrete 2D-Texturen

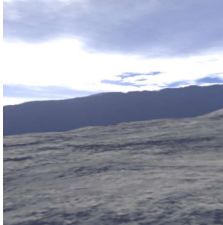
- **Vorteile:**
 - Vorrat an Bildern nahezu unerschöpflich
 - Erzeugung ist einfach (z.B. Photographie)
 - Anwendung auf eine Oberfläche ist sehr schnell
- **Nachteile:**
 - Kontext (Sonnenstand, Schattenwurf, etc.) stimmt meist nicht
 - Bilder hoher Auflösung haben großen Speicherbedarf
 - Fortsetzung meist sehr kompliziert
 - Beim Vergrößern und Verkleinern treten Artefakte auf
 - Verzerrung beim Mapping auf beliebige Fläche

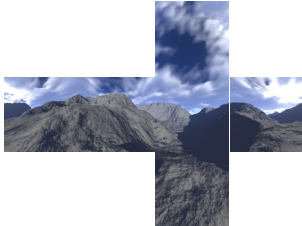
G. Zachmann Computer-Graphik 2 - SS 07 Texturen 15


Beispiel 1: Skybox

- Die Umgebung einer virtuellen Szenen modelliert man oft durch eine Kugel oder einen Würfel mit entsprechenden Texturen






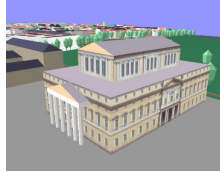




Die Skybox



Vom Boden aus

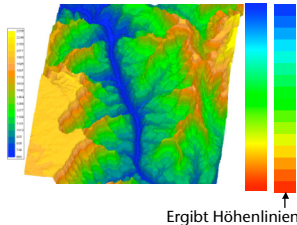


Ohne Skybox

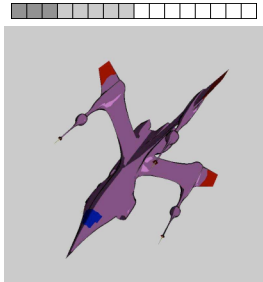
G. Zachmann Computer-Graphik 2 - SS 07 Texturen 16

1D Texturen

- In der Visualisierung möchte man oft einen Parameter durch **Falschfarbendarstellung** intuitiv erfassbar machen
 - z.B. Höhe auf einem Terrain, Temperatur ...
 - Verwende dazu eine 1D-Textur mit einer Farbskala
 - Parameter (z.B. Höhe = y-Koord.) → 1D-Textur-Koord.
- Toon Shading:
 - Berechne Punktprodukt des Licht- und des Normalenvektor oder das Punktprodukt der View- und des Normalenvektors
 - Verwende das als Index in die Farbtabelle (1D-Textur)



Ergibt Höhenlinien



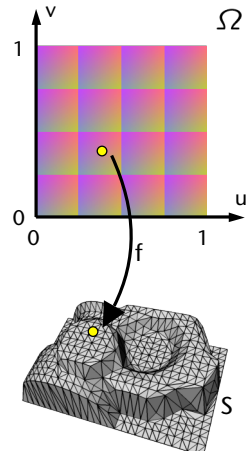
G. Zachmann Computer-Graphik 2 - SS 07 Texturen 17

Das Mapping / die Parametrisierung

- Zu texturierendes Objekt $S =$ Dreiecksnetz
- Textur** :=
 - Parameterraum Ω
 - Pixelbild oder Funktion (diskret / prozedural)
 - Parametrisierung** = Abbildung f zwischen Textur und Objekt:
$$f : \Omega \leftrightarrow S$$
- Texturierung** ist ein 2-stufiger Prozeß
 - Inverses Mapping:

$$(u, v) = f^{-1}(x, y, z)$$
 - Farbe:

$$(r, g, b) = C_{\text{tex}}(u, v)$$



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 18

Texturkoordinaten

- Texturierung eines kompletten Dreiecksnetzes:



- Für jeden Eckpunkt müssen zusätzlich Texturkoordinaten definiert werden, die angeben, welcher Ausschnitt aus der Textur auf das Polygon gemappt wird

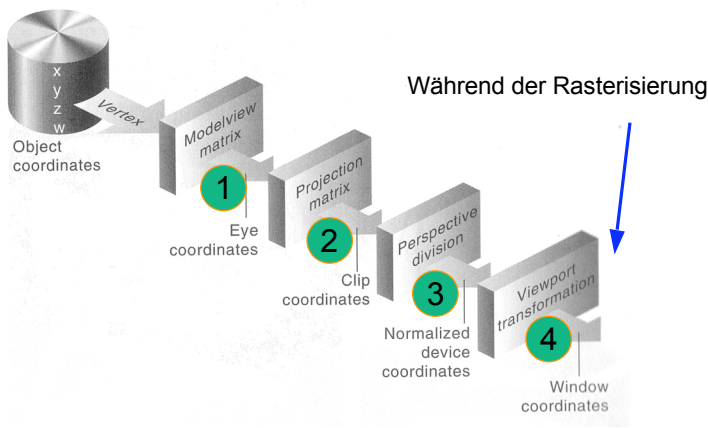



```

glBegin ( GL_... )
  glTexCoord2f (...);
  glNormal3f (...);
  glVertex3f (...);
  ...
glEnd ();
    
```

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 19

Wo wird texturiert?

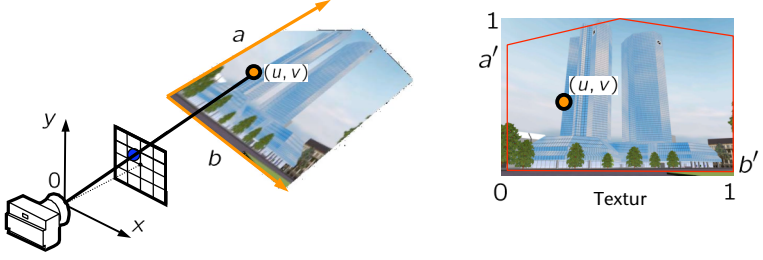


Während der Rasterisierung

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 20

Interpolation der Texturkoordinaten

- Bei der Rasterisierung wird für jedes Pixel die 2D-Oberflächenkoordinate (u,v) ermittelt.
- Diese bestimmt im Koordinatensystem der Textur den Punkt (Texel = "texture element"), der auf das Pixel gemapt wird.



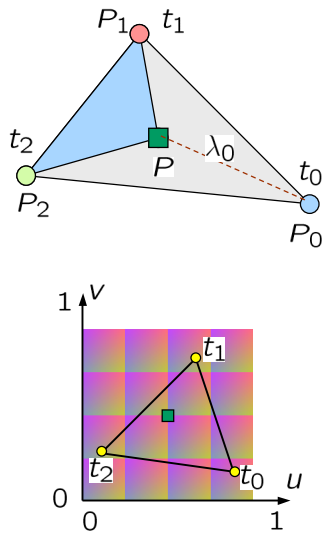
G. Zachmann Computer-Graphik 2 - SS 07 Texturen 21

Generierung der Textur-Koordinaten pro Fragment im Rasterizer

- Baryzentrische Koordinaten

$$\lambda_i(P) = \frac{A(P, P_{i-1}, P_{i+1})}{A(P_0, P_1, P_2)}$$

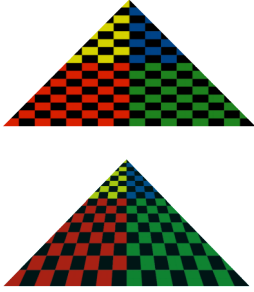

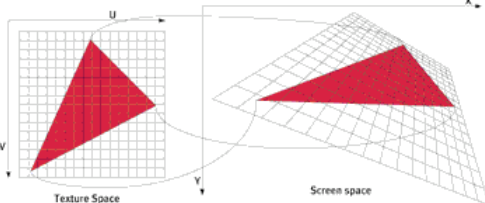
- Gewichte für lineare Interpolation

$$t(P) = \sum_{i=0}^2 \lambda_i(P) t_i$$


G. Zachmann Computer-Graphik 2 - SS 07 Texturen 22

Perspektivische Korrektur

- Problem: bei dieser einfachen, linearen Interpolation im Screen Space entstehen perspektivisch inkorrekte Bilder!
- Ziel: perspektivisch korrekte Interpolation
- Problem: der Rasterizer hat die Koordinaten nur nach der perspektivischen Division!

Demo (mehr auf der VL-Homepage)

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 23

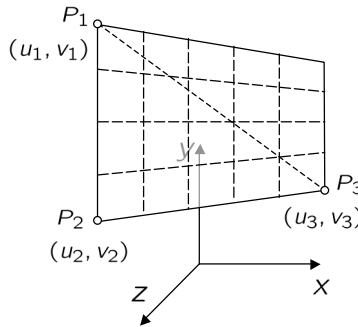
Perspektivische Korrektur

- Erinnerung: was passiert bei der perspektivischen Proj.:

$$P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} x_i \\ y_i \\ z_i \\ w_i \end{pmatrix} \equiv \begin{pmatrix} x_i/w_i \\ y_i/w_i \\ z_i/w_i \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} x_i/w_i \\ y_i/w_i \end{pmatrix} = \hat{P}_i$$

wobei $w_i = \frac{z_i}{z_0}$,
 $z_0 = \text{Proj.ebene}$

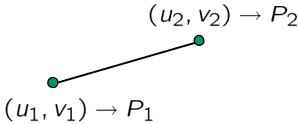
- Erinnerung: baryzentrische Koord. auf dem Rand des Dreiecks = lineare Interpolation zwischen den beiden Ecken



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 24

- Betrachte im Folgenden nur die Interpolation auf einer Linie
- Gegeben: ein t zur linearen Interpolation zwischen \hat{P}_1 und \hat{P}_2 , d.h.

$$\hat{P}(t) = t\hat{P}_1 + (1 - t)\hat{P}_2$$
- Gesucht: Funktionen f_1, f_2 (möglichst ähnlich zu linearer Interpolation), so daß

$$\begin{pmatrix} u \\ v \end{pmatrix}(t) = f_1(t) \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} + f_2(t) \begin{pmatrix} u_2 \\ v_2 \end{pmatrix}$$
 die "richtigen" Texturkoordinaten sind
 

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 25

- Problem:

$$P(t) = tP_1 + (1 - t)P_2 \quad t \in [0, 1]$$

$$\hat{P}(s) = s\hat{P}_1 + (1 - s)\hat{P}_2 \quad s \in [0, 1], \hat{P}_i = \text{Proj}(P_i)$$
 ergeben zwar dieselbe Gerade auf dem Bildschirm, wenn $P(t)$ projiziert wird, aber i.A. ist

$$\text{Proj}(P(t)) \neq \hat{P}(s) \quad \text{obwohl } s = t \quad !$$
- Frage: wie sieht $\text{Proj}(P(t))$ aus?

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 26

- Gegeben:

$$P(t) = t \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + (1-t) \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

- O.B.d.A. betrachte nur die x-Koordinate:

$$x(t) = tx_2 + (1-t)x_1 \mapsto \frac{tx_2 + (1-t)x_1}{tw_2 + (1-t)w_1}$$

$$\text{wobei } w_i = \frac{z_i}{z_0}$$

- Behauptung:

$$\frac{tx_2 + (1-t)x_1}{tw_2 + (1-t)w_1} = \frac{x_1}{w_1} + \frac{tw_2}{w_1 + t(w_2 - w_1)} \left(\frac{x_2}{w_2} - \frac{x_1}{w_1} \right)$$

- Beweis:

$$\frac{x_1}{w_1} + \frac{tw_2}{w_1 + t(w_2 - w_1)} \left(\frac{x_2}{w_2} - \frac{x_1}{w_1} \right) =$$

$$\frac{x_1(w_1 + t(w_2 - w_1)) + tw_2w_1 \left(\frac{x_2}{w_2} - \frac{x_1}{w_1} \right)}{w_1(w_1 + t(w_2 - w_1))} =$$

$$\frac{x_1w_1 + tw_2x_1 - tw_1x_1 + tw_1x_2 - tw_2x_1}{w_1^2 + tw_2w_1 - tw_1^2} =$$

$$\frac{x_1w_1 - tw_1x_1 + tw_1x_2}{w_1^2 + tw_2w_1 - tw_1^2} = \frac{x_1 - tx_1 + tx_2}{w_1 + tw_2 - tw_1} =$$

$$\frac{x_1 + t(x_2 - x_1)}{w_1 + t(w_2 - w_1)} = \frac{tx_2 + (1-t)x_1}{tw_2 + (1-t)w_1}$$

- Gegeben:
$$\hat{P}(s) = s \begin{pmatrix} \hat{x}_2 \\ \hat{y}_2 \end{pmatrix} + (1 - s) \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix}$$
- Frage: welches t passt zu diesem s , d.h., für welches t ist
$$\text{Proj}(P(t)) = \hat{P}(s)$$

$$s\hat{x}_2 + (1 - s)\hat{x}_1 = \frac{x_1}{w_1} + s\left(\frac{x_2}{w_2} - \frac{x_1}{w_1}\right)$$

$$\Rightarrow s = \frac{tw_2}{w_1 + t(w_2 - w_1)}$$

$$\Rightarrow t = \frac{sw_1}{w_2 + s(w_1 - w_2)}$$
- Mit diesem t kann man die Texturkoord. u, v linear interpolieren!



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 30

Beeinflussung der Beleuchtung

- Wie kann ein Texturwert die Beleuchtungsrechnung beeinflussen, was kann man mit einer Textur machen?
- Erinnerung: Blinn-Phong Modell

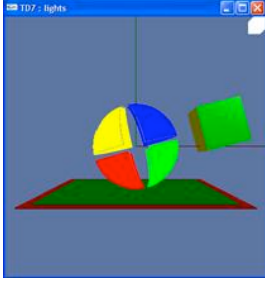
$$L_{\text{Phong}} = r_a L_a + \sum_j (r_d(\mathbf{n} \cdot \mathbf{l}_j) + r_s(\mathbf{h}_j \cdot \mathbf{e})^m) L_j$$

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 32

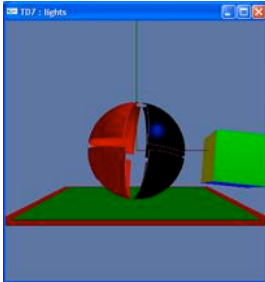
1. Ersetzen der Objektfarbe (*replace*)

- Einfachste Art der Texturierung
- Jegliche Beleuchtung wird entfernt

$$L_{\text{out}} = C_{\text{tex}}(u, v)$$




2. A posteriori Skalierung der Farbe (*modulate*)

- Häufigste Art der Texturierung
- Komponentenweise Skalierung des Farbwertes

$$L_{\text{out}} = L_{\text{Phong}} \cdot C_{\text{tex}}(u, v)$$


G. Zachmann Computer-Graphik 2 - SS 07

Texturen 33

3. A priori Skalierung der Materialfarbe

- Farbe des Objektes wird im wesentlichen durch r_a und r_d bestimmt

$$r_a = k_a \cdot C_{\text{tex}}(u, v) \quad r_d = k_d \cdot C_{\text{tex}}(u, v)$$


- Wichtig: im Unterschied zu 2 bleibt der spekulare Anteil von der Textur unbeeinflusst
- In OpenGL mittels `GL_SEPARATE_SPECULAR` erreichbar

4. Modulation der spekularen Reflexion (*gloss mapping*)

- Analog zu 3 für r_s

$$r_s = k_s \cdot C_{\text{tex}}(u, v)$$

- Erlaubt Modellierung unregelmäßiger "shininess" (z.B. verschmutzte Flächen)
- Geht nur mit Vertex- und Fragment-Shaders



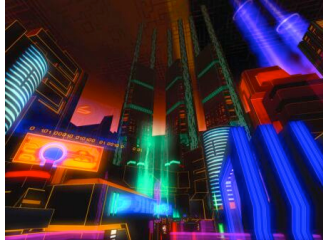
G. Zachmann Computer-Graphik 2 - SS 07

Texturen 34

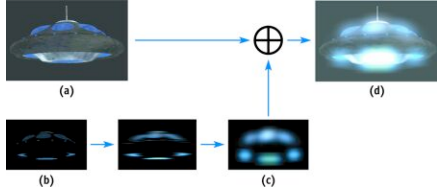
4.1 "Glow"-Effekt:

$$L_{\text{out}} = C_{\text{tex}}(u, v) + L_{\text{Phong}}$$

- Für neon signs, TV, laser beams etc.



- Geht (vermutlich) nur mit Multi-Pass-Rendering




G. Zachmann Computer-Graphik 2 - SS 07 Texturen 35

5. Modulation der Transparenz

- Speichern der „Durchsichtigkeit“ in einer Textur:

$$\alpha = \alpha_{\text{obj}} \cdot \alpha_{\text{tex}}(u, v)$$
- Pixel mit $\alpha=0$ sind voll durchsichtig und Pixel mit $\alpha=1$ sind voll undurchsichtig
- Ermöglicht komplexe Umriss

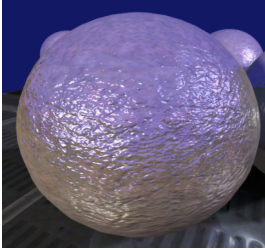




G. Zachmann Computer-Graphik 2 - SS 07 Texturen 36

6. Perturbation der Normale (*Bump- / Normal-Mapping*)

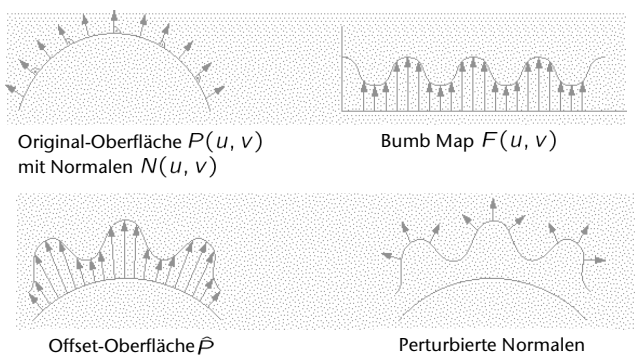
- Speichern von Höhenwerten oder Normalen einer Offsetfläche in einer Textur

$$\mathbf{n} = f(C_{\text{tex}}(u, v))$$


G. Zachmann Computer-Graphik 2 - SS 07 Texturen 37

Bump Mapping (ursprüngliche Idee)

- Bump-Map = skalare Textur
- Resultierende Oberfläche:



Original-Oberfläche $P(u, v)$ mit Normalen $N(u, v)$

Bump Map $F(u, v)$

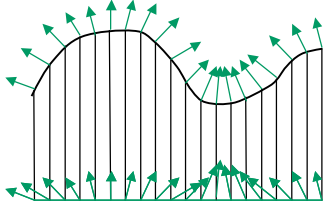
Offset-Oberfläche \hat{P}

Perturbierte Normalen

$$\hat{P}(u, v) = P(u, v) + F(u, v) \frac{N(u, v)}{\|N(u, v)\|}$$

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 38

- Beobachtung: ins Beleuchtungsmodell geht **nicht direkt** $P(u, v)$, sondern **nur** $N(u, v)$ ein.
- Hauptidee des Bump-Mapping: für kleine Unebenheiten genügt Visualisierung von $P(u, v)$ mit $\hat{N}(u, v)$ aus



- Wie berechnet man $\hat{N}(u, v)$:

$$\hat{N}(u, v) = \hat{P}_u(u, v) \times \hat{P}_v(u, v)$$

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 39

- Richtungsableitungen mit Summen- und Kettenregeln:

$$\hat{P}_u(u, v) = P_u(u, v) + F_u(u, v) \frac{N(u, v)}{\|N(u, v)\|} + F(u, v) \frac{d}{du} \frac{N(u, v)}{\|N(u, v)\|}$$

$$\hat{P}_v(u, v) = P_v(u, v) + F_v(u, v) \frac{N(u, v)}{\|N(u, v)\|} + F(u, v) \frac{d}{dv} \frac{N(u, v)}{\|N(u, v)\|}$$

- Falls $F(u, v)$ klein \rightarrow Weglassen des letzten Teilterms:

$$\hat{P}_u(u, v) \approx P_u(u, v) + F_u(u, v) \frac{N(u, v)}{\|N(u, v)\|}$$

$$\hat{P}_v(u, v) \approx P_v(u, v) + F_v(u, v) \frac{N(u, v)}{\|N(u, v)\|}$$

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 40

- Für $\hat{N}(u, v)$ folgt damit:

$$\begin{aligned}
 \hat{N} &= \hat{P}_u \times \hat{P}_v \\
 &= P_u \times P_v + F_u \left(\frac{N}{\|N\|} \times P_v \right) + F_v \left(P_u \times \frac{N}{\|N\|} \right) + F_u F_v \left(\frac{N}{\|N\|} \times \frac{N}{\|N\|} \right) \\
 &= P_u \times P_v + F_u \left(\frac{N}{\|N\|} \times P_v \right) + F_v \left(P_u \times \frac{N}{\|N\|} \right) \\
 &= N + \frac{1}{\|N\|} (F_u(N \times P_v) - F_v(N \times P_u))
 \end{aligned}$$

- Die Ableitungen F_u und F_v können mit finiten Differenzen approximiert werden.
- Finite Differenzen auf uniformem Gitter der Gittergröße h (im 1D)

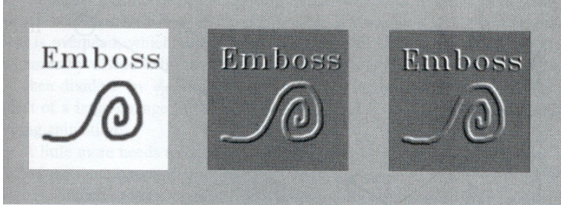
- Vorwärtsdifferenzen $f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$

- Rückwärtsdifferenzen $f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$

- Zentrale Differenzen $f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1}))}{2h}$

Bemerkungen

- Speicherung:
 - Höhenfeld als Grauwertbild in R (z.B. mit Malprogramm erstellt)
 - Richtungsableitungen (mit finiten Differenzen berechnet) in G/B speichern

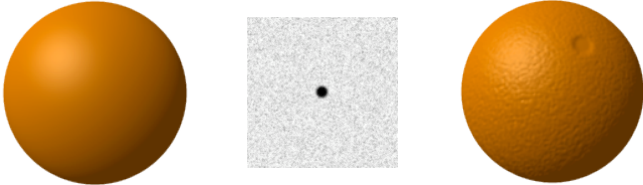


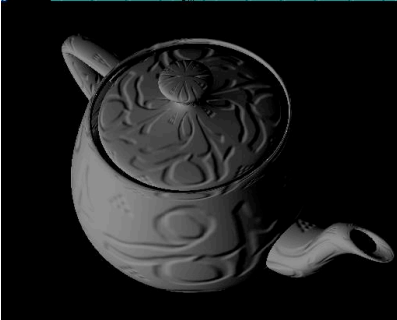
Original Höhenfeld u-Richtungsableitung v-Richtungsableitung


- Voraussetzung: Beleuchtung erst bei der Rasterisierung, oder sehr fein tesselierte Geometrie und dann Berechnung der Normalen an jedem Vertex "von Hand"

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 43

Weitere Beispiele





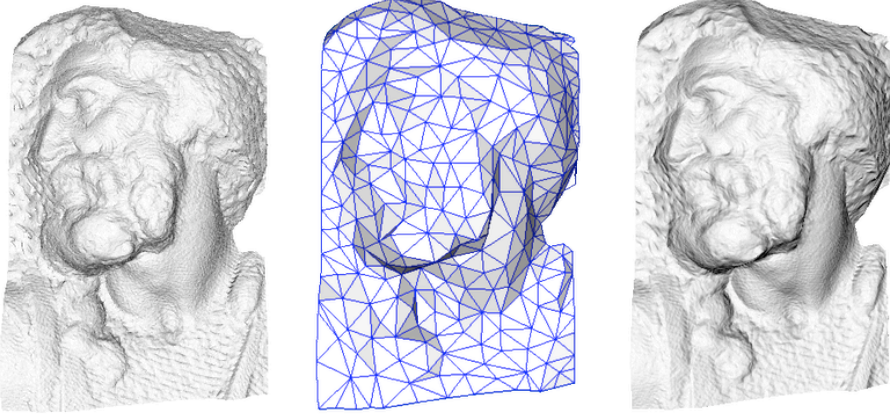


Multi-Textures (Bump und Environment)

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 44

Normal Mapping

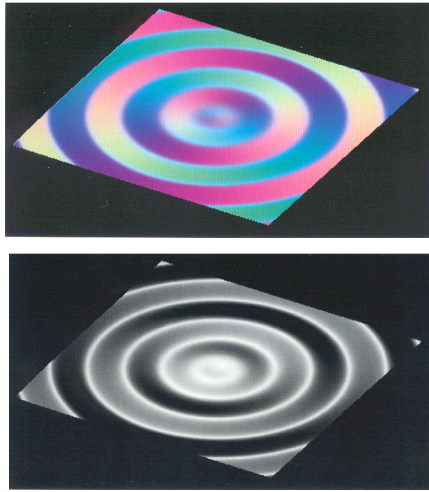
- Normalen in hoher Auflösung
- Für niedrig aufgelöste Geometrie



The image shows three stages of normal mapping. On the left is a high-resolution normal map of a bust, showing fine detail in grayscale. In the middle is a low-resolution mesh of the same bust, represented by a blue wireframe. On the right is the final rendered result, where the high-resolution normal map is applied to the low-resolution mesh, creating a detailed appearance.

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 45

Beispiel

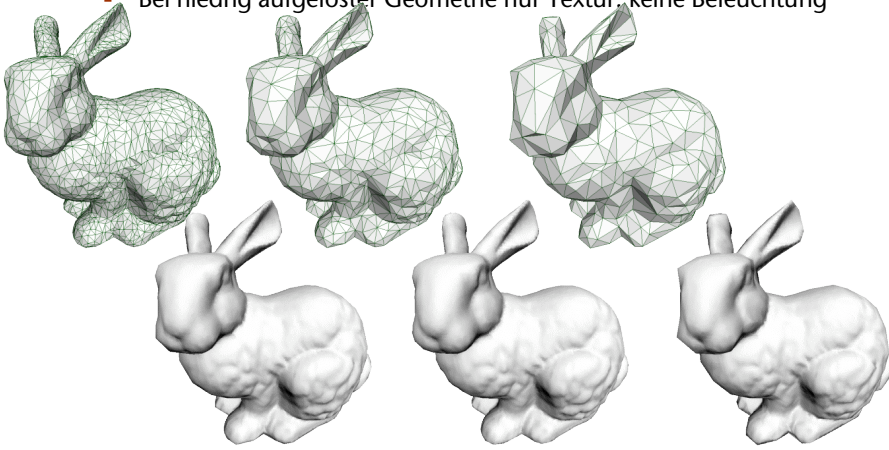


The image shows two examples of normal maps. The top one is a colorful normal map with a rainbow gradient, and the bottom one is a grayscale normal map with a similar gradient. Both are applied to a low-resolution mesh of a square with a circular hole.

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 46

7. Shading-Texturen:


- Shading in hoher Auflösung → Textur
- Bei niedrig aufgelöster Geometrie nur Textur, keine Beleuchtung



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 48

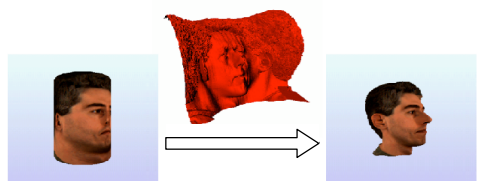
8. *Displacement Mapping* (Offsetflächen):

- Tatsächliche Veränderung der Geometrie (zusätzlich zur Normalenperturbation)



Geometrie Bump Mapping Displacement Mapping

- Im Gegensatz zu Bump-Mapping auch größerer Offset möglich



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 49

- Durch folgende Komponenten beschrieben:
 - Skalarfeld: $D : (u, v) \rightarrow D(u, v)$
 - Normalenfeld: $\mathbf{n}' : (u, v) \rightarrow \mathbf{n}'(u, v)$

displaced surface

base surface


base triangle

- Voraussetzung: Beleuchtung bei der Rasterisierung (vor Depth-Buffer)


G. Zachmann Computer-Graphik 2 - SS 07
Texturen 50


Beispiel

G. Zachmann Computer-Graphik 2 - SS 07
Texturen 51




- Light Maps:
 - Zusätzlicher Textur wird verwendet, um statische oder dynamische Illumination zur Szene hinzuzufügen


 - Weil Illumination räumlich nur niedrige Frequenzen hat, ist nur eine gering aufgelöste Textur erforderlich
 - Viele kleine Light Maps können in eine große Textur verpackt werden
 - Light Maps werden gewöhnlich mittels Raytracing oder Radiosity erzeugt



G. Zachmann Computer-Graphik 2 - SS 07 Texturen 52



Texturen in Open GL

- Als erstes muss eine Textur definiert 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** = $2^n + 2 * \text{border}$ 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

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 53

- Textur einschalten:


```
glEnable( GL_TEXTURE_{12}D )
```
- Zu jedem Eckpunkt gehört eine Texturkoordinate:


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

 - das Bild liegt dabei im Bereich $[0,1] \times [0,1]$
 - 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

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 54

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 ...

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 55

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

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 56

Beeinflussung der Pixelfarbe in OpenGL

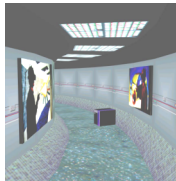
- Funktion:
`glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, value)`
- 4 Möglichkeiten für `value`:
 - `GL_REPLACE`: Texelfarbe ersetzt Pixelfarbe (am häufigsten)
 - `GL_MODULATE`: komponentenweise Mult. von T und F

$$T_{RGB} \cdot F_{RGB}$$
 - `GL_DECAL`:

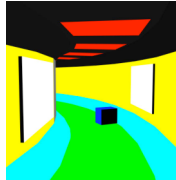
$$\alpha_T \cdot T_{RGB\alpha} + (1 - \alpha_T) \cdot F_{RGB\alpha}$$
 - `GL_BLEND`:

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

C wird definiert über
`glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, value)`



T = Texelfarbe



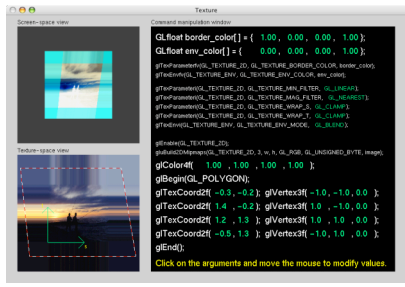
F = Pixelfarbe ohne Textur

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 57

Koordinaten-Wrap

- Was geschieht, wenn Texturkoordinaten außerhalb $[0,1] \times [0,1]$ 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
```



```
GLfloat border_color[] = { 1.00, 0.00, 0.00, 1.00 };
GLfloat env_color[] = { 0.00, 0.00, 0.00, 1.00 };
GLfloat border_color[] = { 1.00, 0.00, 0.00, 1.00 };
GLfloat env_color[] = { 0.00, 0.00, 0.00, 1.00 };
GLuint texture_id;
glGenTextures( 1, &texture_id );
glBindTexture( GL_TEXTURE_2D, texture_id );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_R, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_R, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glColor4f( 1.00, 1.00, 1.00, 1.00 );
glBegin( GL_POLYGON );
glTexCoord2f( -0.3, -0.2 ); glVertex3f( -1.0, -1.0, 0.0 );
glTexCoord2f( 1.4, -0.2 ); glVertex3f( 1.0, -1.0, 0.0 );
glTexCoord2f( 1.2, 1.3 ); glVertex3f( 1.0, 1.0, 0.0 );
glTexCoord2f( -0.5, 1.3 ); glVertex3f( -1.0, 1.0, 0.0 );
glEnd();
```

```
cd nate_robbins_tutors;
./texture
```

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 58

Textur-IDs

- Während des Renderings einer Szene benötigt man viele verschiedene Texturen
- Jedesmal `glTexImage2D()` ist ineffizient
- 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 )
```

 dadurch werden alle Textur-relevanten Teile des Zustandes umgeschaltet!

G. Zachmann Computer-Graphik 2 - SS 07 Texturen 59

▪ 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();

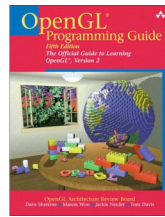
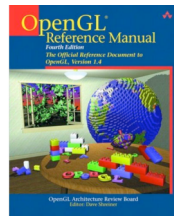
```



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/>