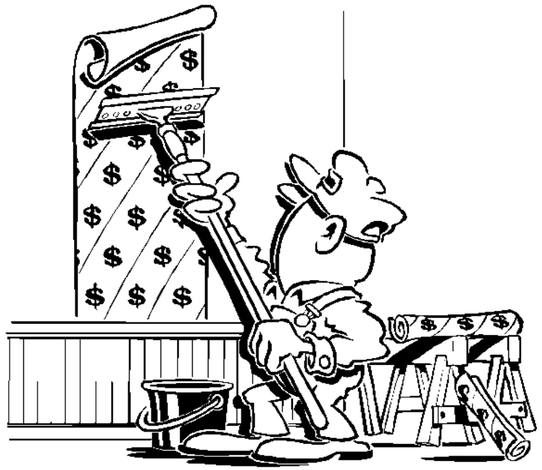
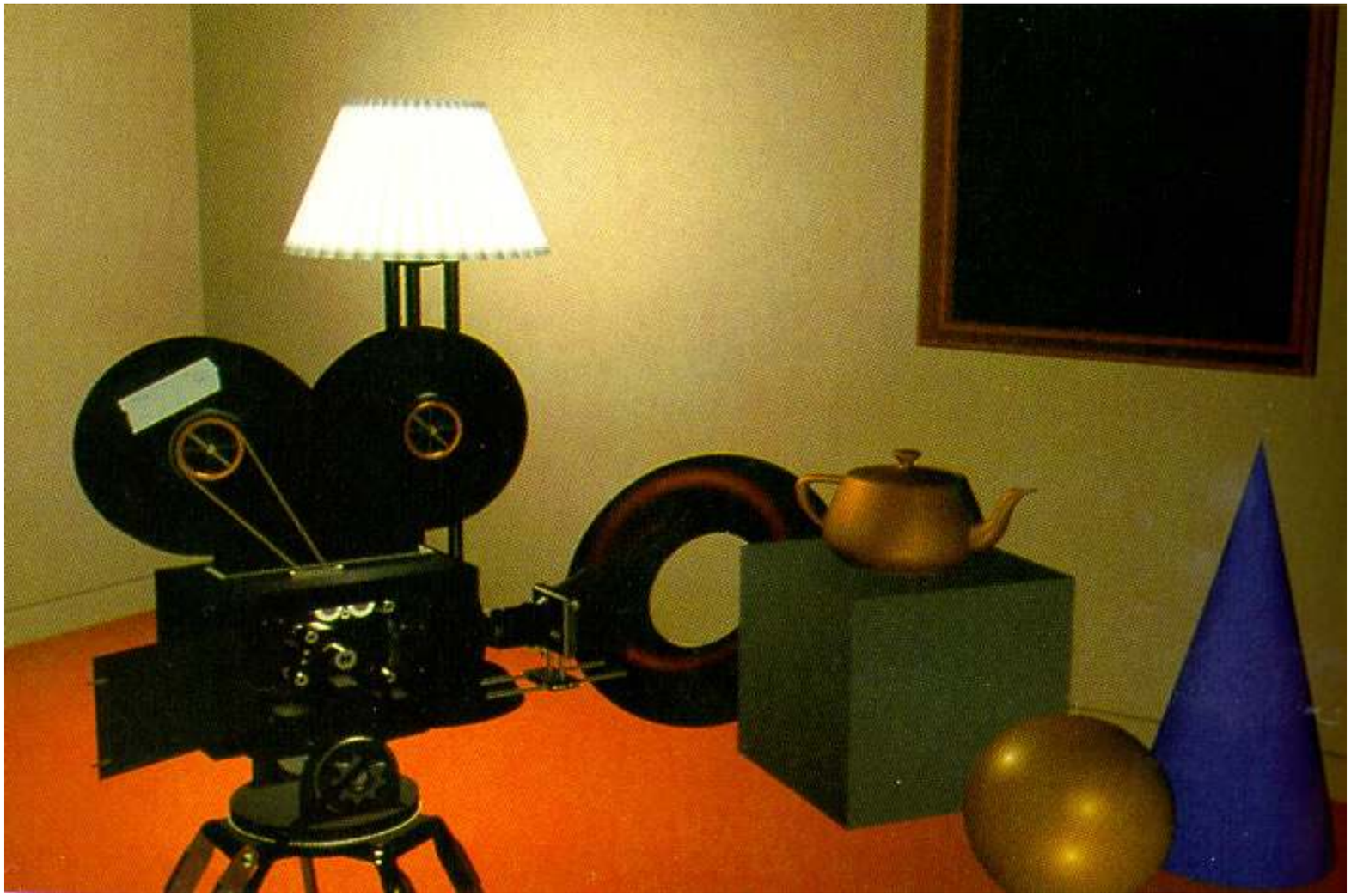
Computer-Graphik I
Texturierung



G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Motivation

Was fehlt?



"Shutter bug", Pixar

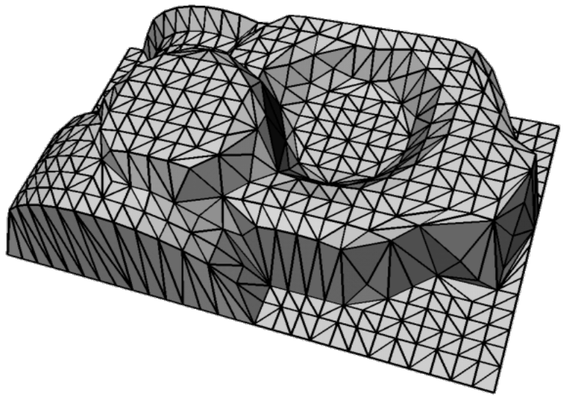
Oberflächendetails

"Shutter bug", Pixar

- Großes Spektrum geometrischer Formen und physikalischer Materialien:
  - Strukturen unebener Oberflächen, z.B. Putzwände, Leder, Schale/Rinde von Orangen, Baumstämme, Maserungen in Holz und Marmor, Tapeten mit Muster, etc.
  - Wolken
  - Objekte im Hintergrund (Häuser, Maschinen, Pflanzen und Personen)
- Solche Objekte durch Flächen nachzubilden ist in der Regel viel zu aufwendig

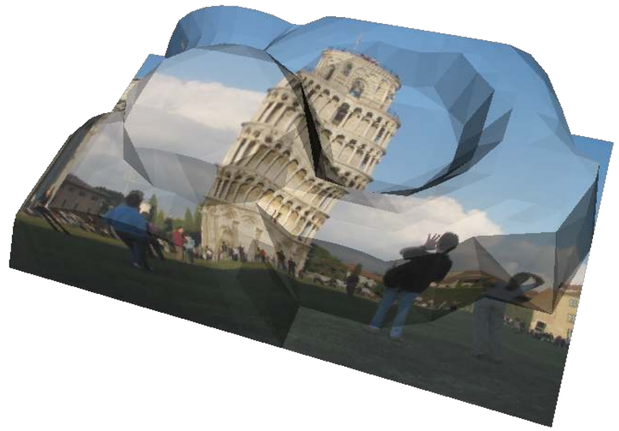
Grundidee der Texturierung

- Ziel: Visuelles Detail trotz grober Geometrie
- Idee: Objekt (grobe Shape) mit Textur (visuelles Detail) "tapezieren"

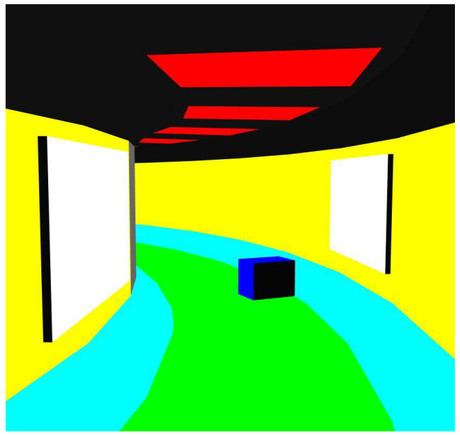


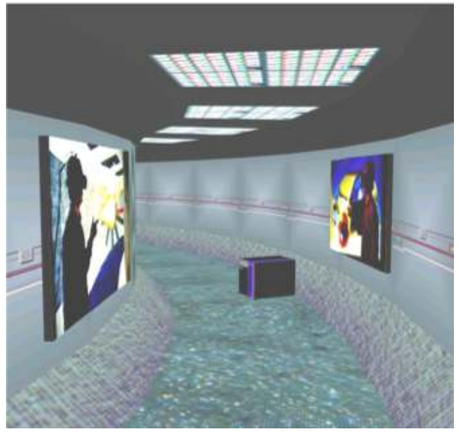
Objekt (Geometrie)

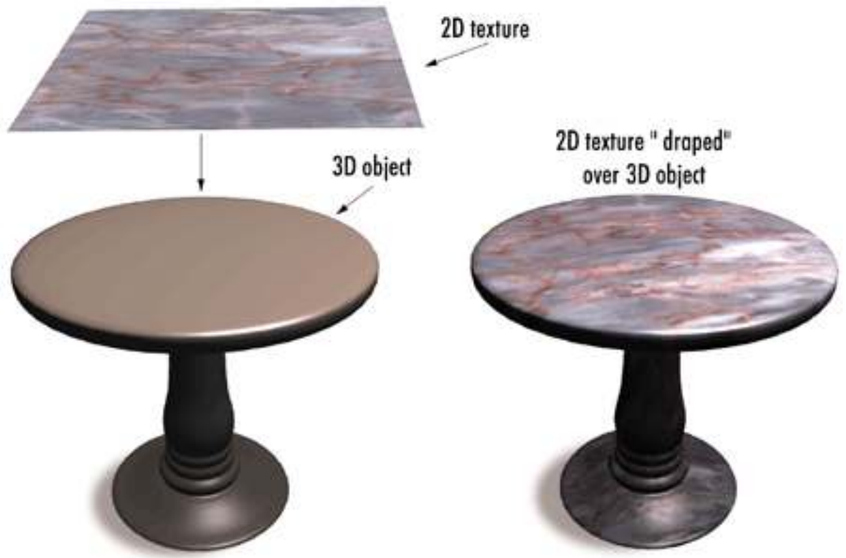
Textur (Farbe)

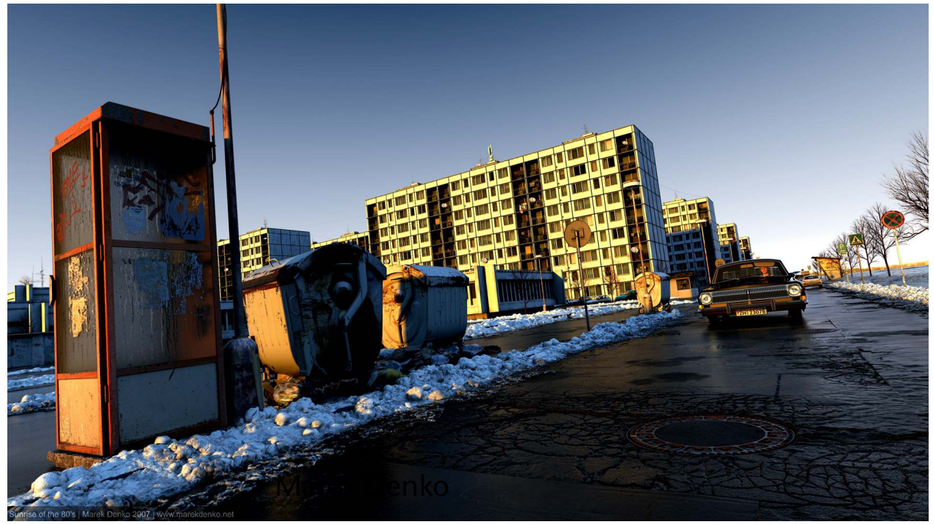


Weitere Beispiele









Eine "Kaustik-Textur" verstärkt den Unterwassereindruck

Übersicht

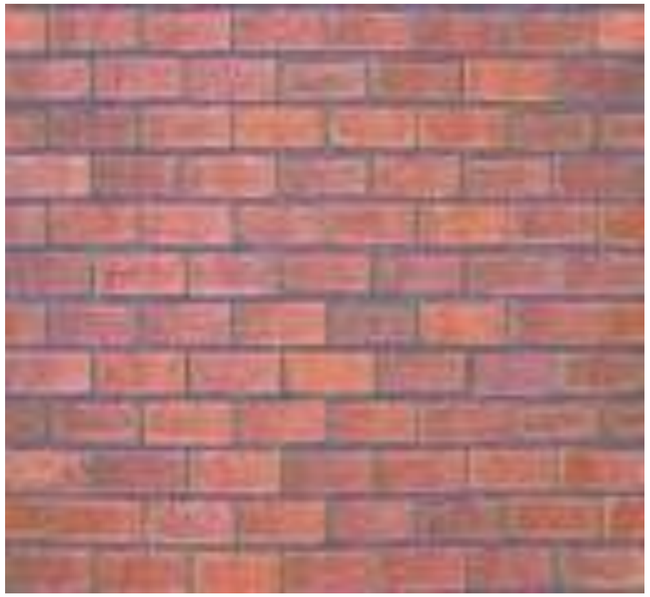
- Kategorien von Texturen: diskret oder prozedural
- Dimension der Texturen: 1D, 2D, 3D, (4D)
- Wichtige Punkte bei den diskreten 2D-Texturen:
  1. Interpolation der Texturkoordinaten
  2. Anwendung der Textur auf die Beleuchtung o. a. Oberflächeneigensch.
  3. Parametrisierung der Fläche
  4. Filterung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

Texturen kommen in verschiedenen Dimensionen

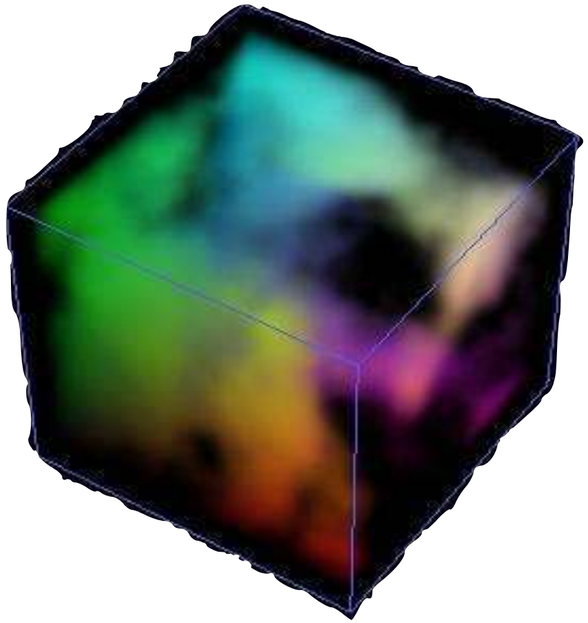
- Textur kann als Funktion einer, zweier oder dreier Koordinaten, oder als Funktion einer Richtung gesehen werden:

1D Textur

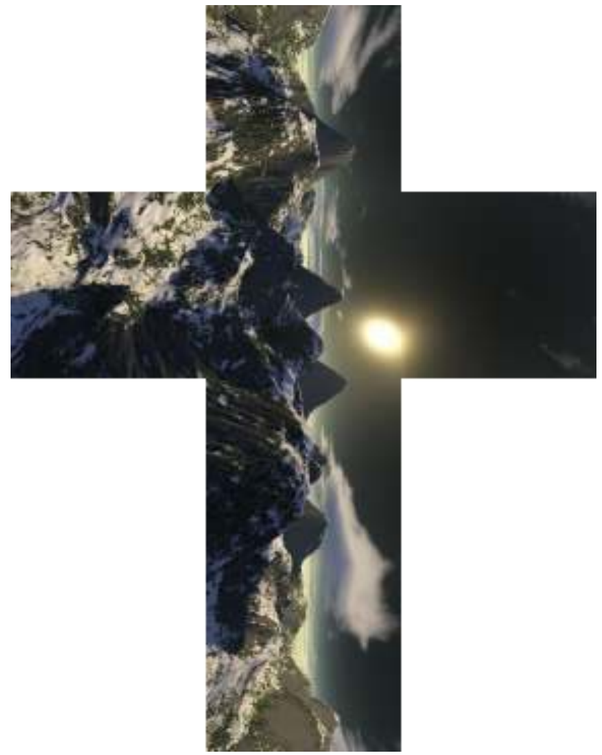
2D Textur



3D Textur



Cubemap Texturen



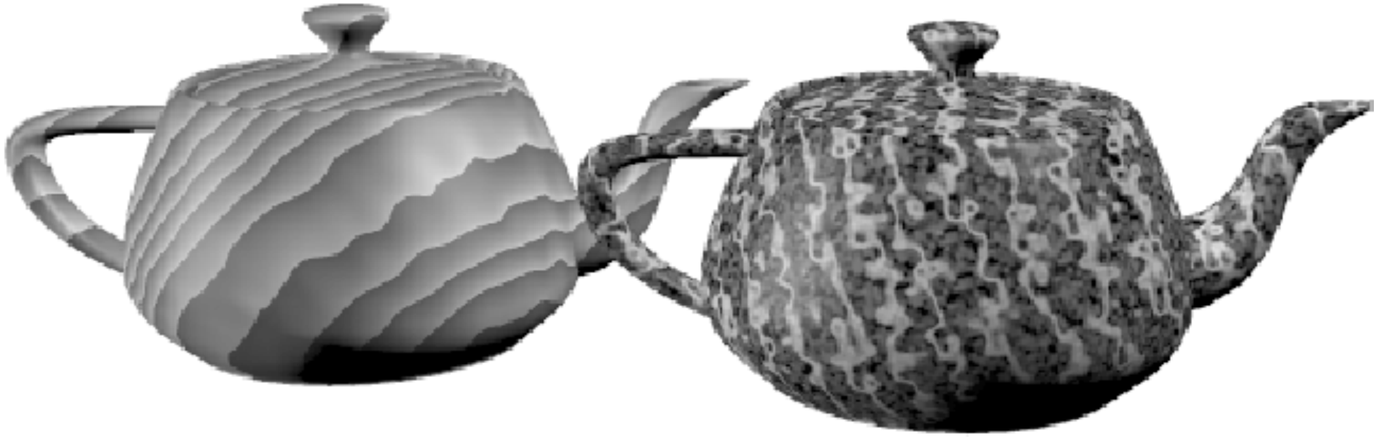
Einfacher Fall: 3D-Texturen

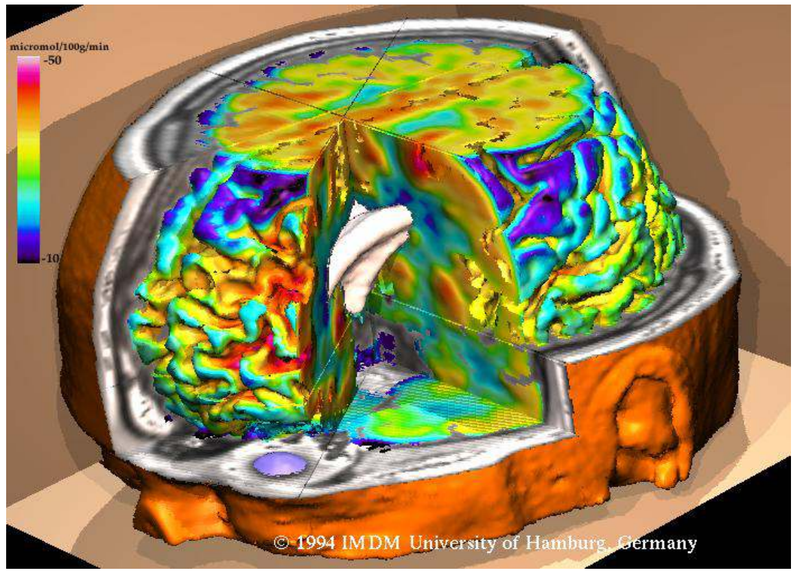
- Die lokalen Koordinaten der Objekt-Oberfläche (*x,y,z*) indizieren direkt die Textur:

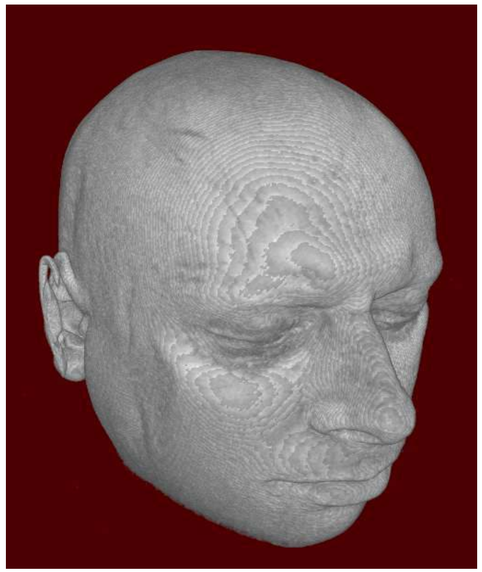
$$(r, g, b) = C_{\text{tex}}(x, y, z)$$

- Die Textur ist also an jedem Punkt im Raum definiert (theoretisch)
- Das Objekt wird quasi aus dem Texturvolumen "herausgeschnitzt"
- 3D-Texturen nennt man auch Festkörper-Texturen (z.B. Holz und Marmor) *("solid texture")*

Beispiele




micromol/100g/min
-50
-10
© 1994 IMDM University of Hamburg, Germany



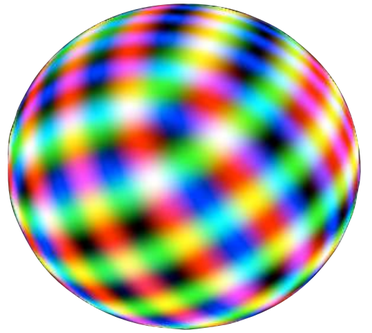
Diskrete und prozedurale Texturen

- Eine diskrete 3D-Textur = 3-dimensionales Array C[u,v,w]
  - C[u,v,w] = Vektor mit 3 Farbkomponenten = ein "Texel" (*texture element*)
  - Pro Pixel benötigt man 3 Texturkoordinaten (*u,v,w*) zum Indizieren in das Array
- Prozedurale Texturen werden an jeder Stelle im Raum aus einer mathematischen Funktion oder einem Algorithmus berechnet:

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

- Einfaches Beispiel für eine prozedurale 3D-Textur:

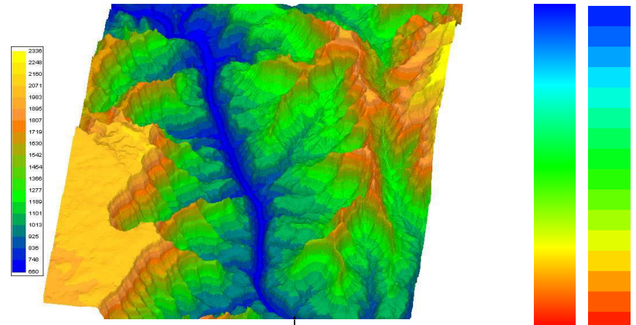
$$C = \begin{pmatrix} \frac{1}{2}(1 + \sin(\frac{\pi}{w_x} P_x)) \\ \frac{1}{2}(1 + \sin(\frac{\pi}{w_y} P_y)) \\ \frac{1}{2}(1 + \sin(\frac{\pi}{w_z} P_z)) \end{pmatrix}$$



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

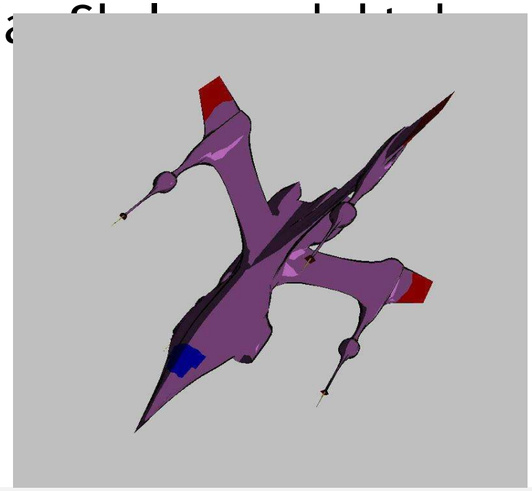
1D Texturen

- 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.



Ergibt Höhenlinien

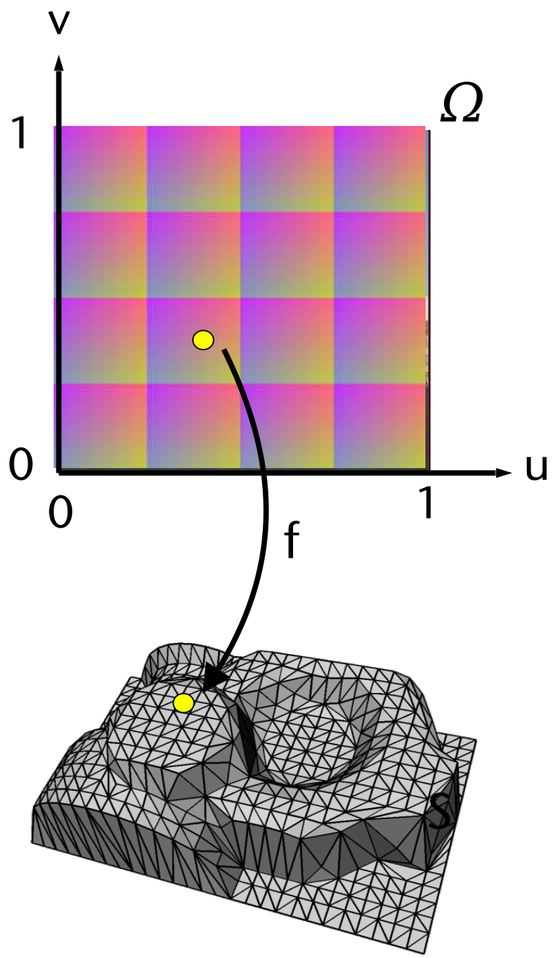
- Toon Shading:
  - Berechne Punktprodukt des Licht- und des Normalenvektor oder da… View- und des Normalenvektors
  - Verwende dieses als Index in die Farbtabelle (1D-Textur)



Formale Definition von 2D-Texturen

- Zu texturierendes Objekt S = Dreiecks-Mesh
- Textur :=
  1. Parameterraum Ω (parametric domain)
  2. Pixelbild oder Funktion (diskret bzw. prozedural)
  3. Parametrisierung / Mapping = Abbildung f zwischen Textur und Objekt:

$$f : \Omega \leftrightarrow S$$



- Texturierung ist ein 2-stufiger Prozeß
  1. Inverses Mapping: $(u, v) = f^{-1}(x, y, z)$
  2. Farbe: $(r, g, b) = C_{tex}(u, v)$

2D-Texturierung

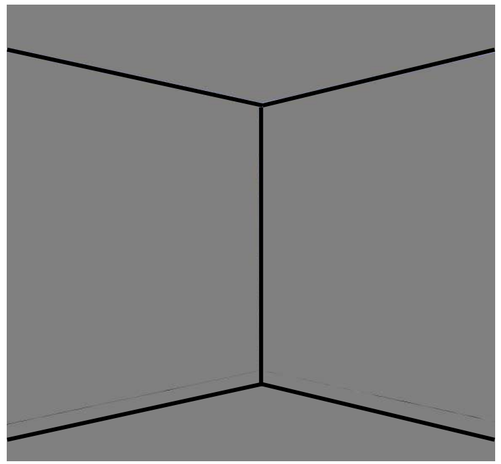
3D-Texturierung

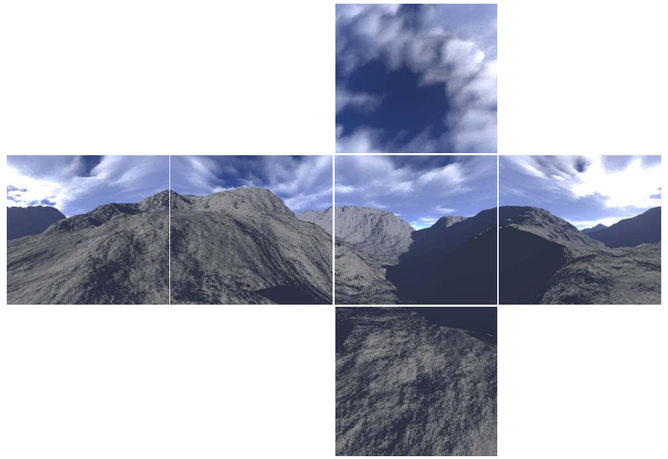
Diskrete 2D-Texturen

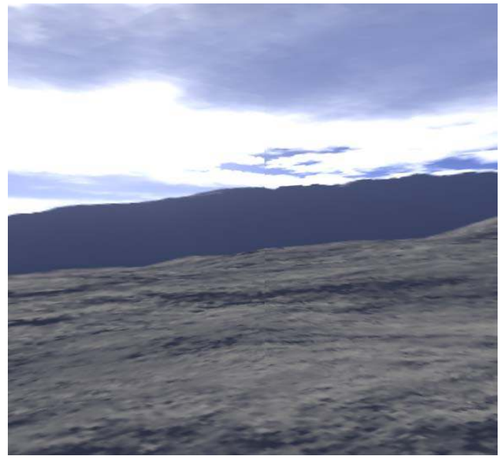
- Vorteile:
  - Vorrat an Bildern nahezu unerschöpflich
  - Erzeugung ist einfach (z.B. Photographie)
  - Anwendung auf eine Oberfläche ist sehr schnell
- Nachteile:
  - Kontext (Sonnenstand, Schattenwurf, etc.) stimmt meist nicht
  - Bilder hoher Auflösung haben großen Speicherbedarf
  - Fortsetzung meist sehr kompliziert
  - Beim Vergrößern und Verkleinern treten Artefakte auf (s. Mipmapping)
  - Verzerrung beim Mapping auf beliebige 3D Oberflächen

Einfaches Beispiel: die *Skybox*

- Die Umgebung einer virtuellen Szene modelliert man oft durch einen Würfel mit entsprechenden Texturen





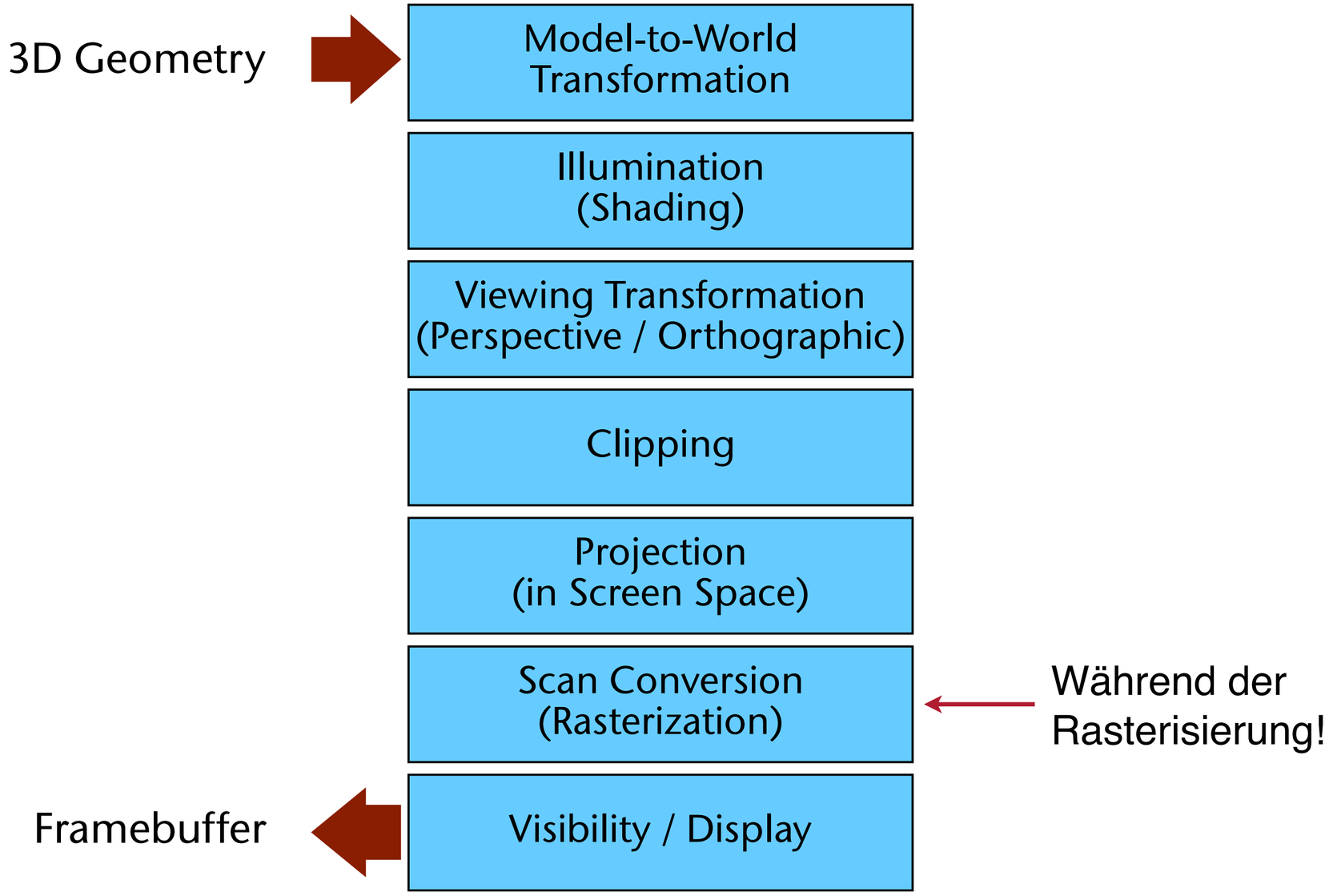


Ohne Skybox

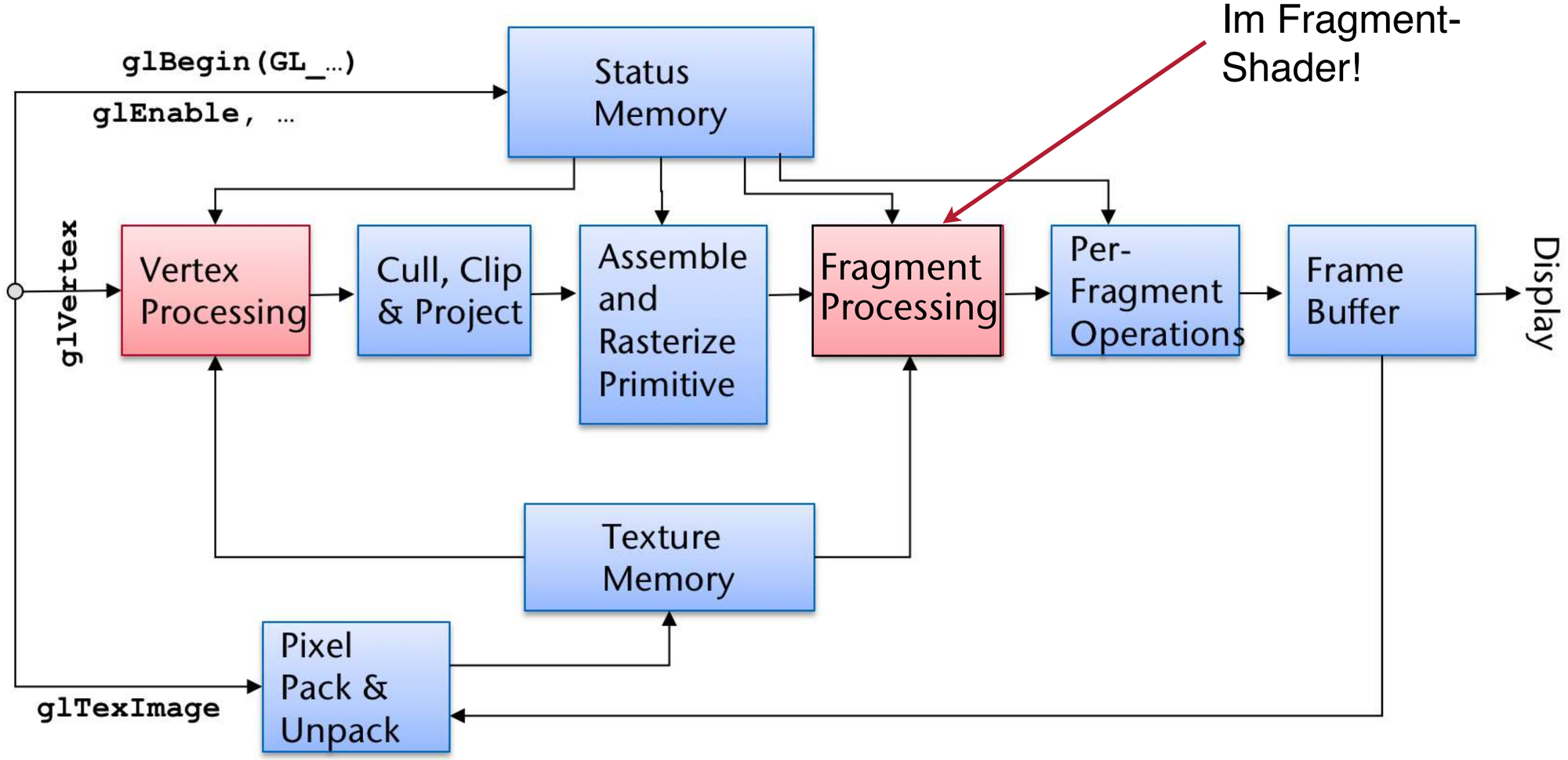
Die Skybox

Vom Boden aus

Wo wird in der (fixed function) Pipeline texturiert?

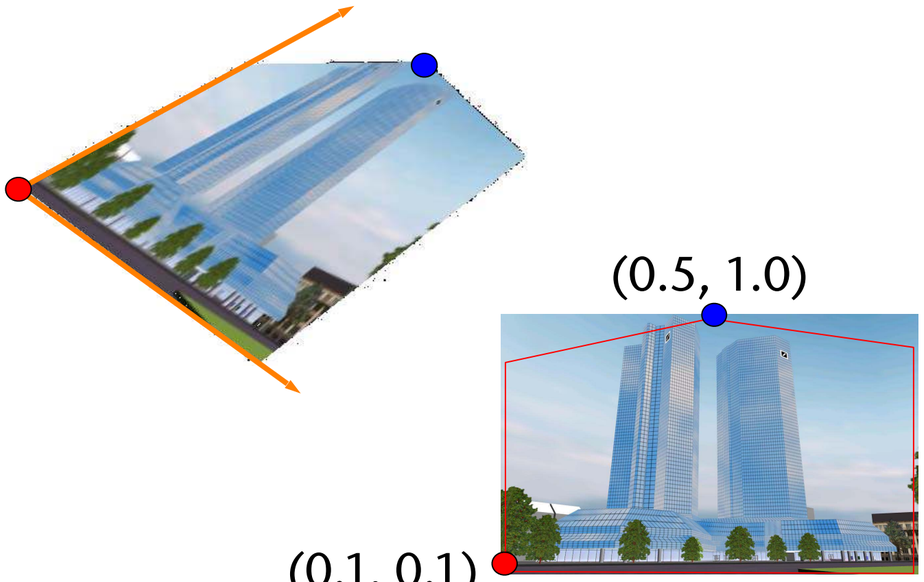


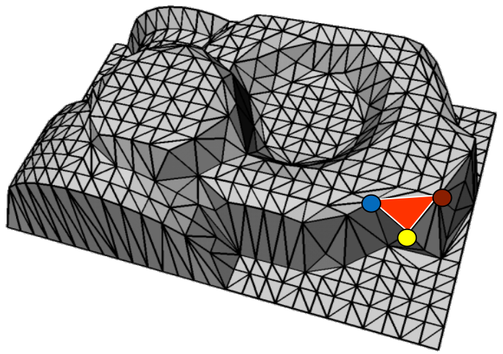
In der programmierbaren Pipeline

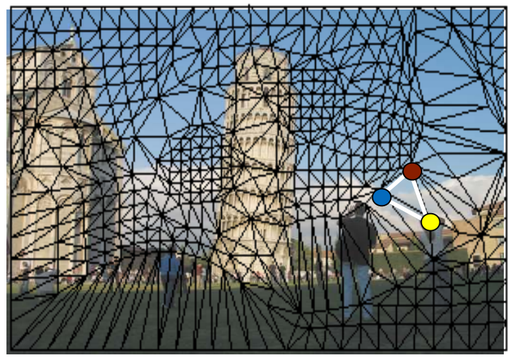


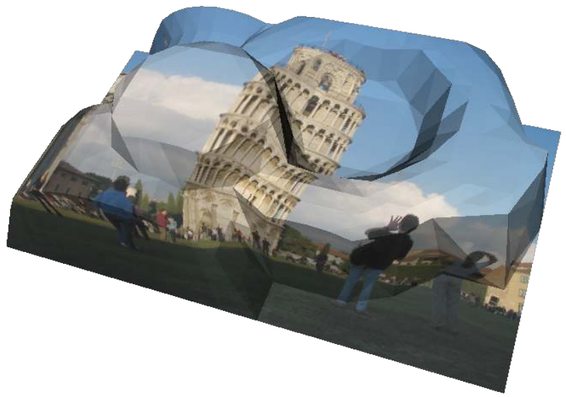
Stückweise lineare Parametrisierung durch Texturkoordinaten

- Für jeden Vertex des Polygon-Meshes müssen zusätzlich Texturkoordinaten definiert (oder berechnet) werden, die angeben, welcher Ausschnitt aus der Textur auf das Polygon gemappt wird
- Texturierung eines kompletten Dreiecks-Mesh durch u,v-Koordinaten



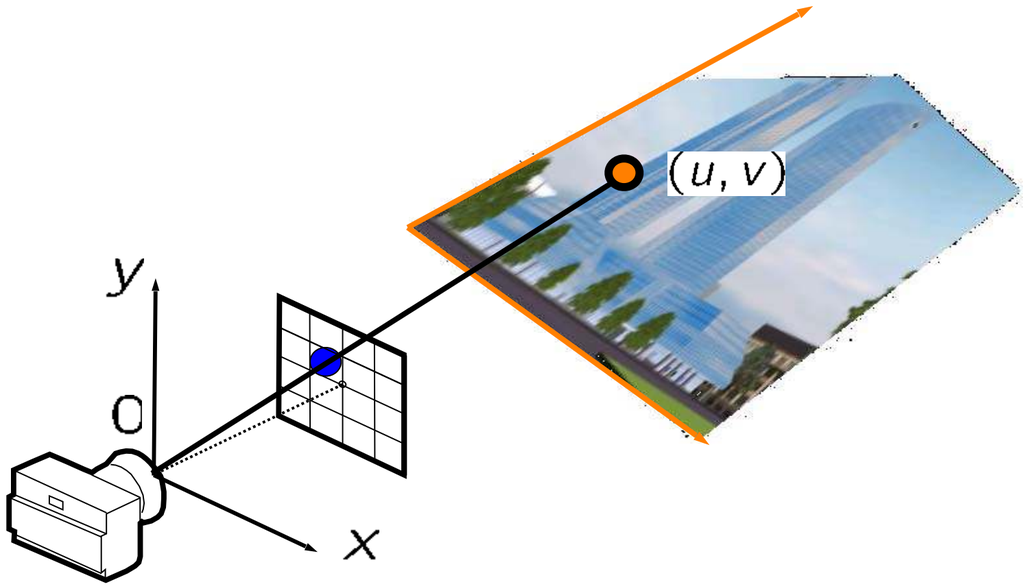


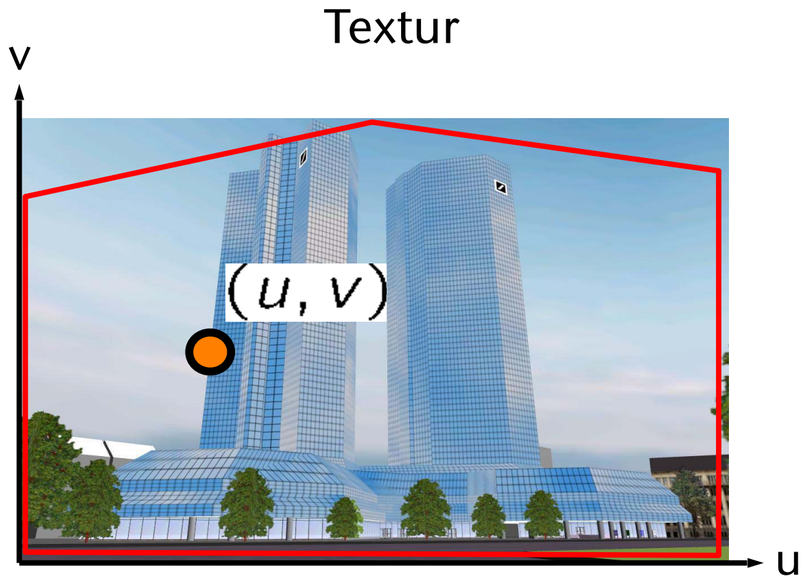




Interpolation der Texturkoordinaten

- Bei der Rasterisierung muss *für jedes Fragment* eine Texturkoordinate (*u*,*v*) aus den Texturkoordinaten der Vertices generiert werden
- Diese bestimmt im Koordinatensystem der Textur das Texel (= Pixel der Textur), das auf das Pixel (im Framebuffer) gemapt wird

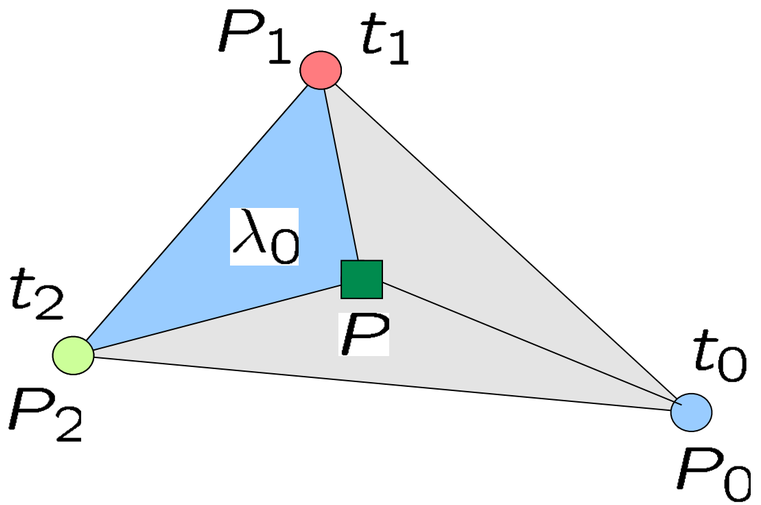




Interpolation der Textur-Koordinaten pro Fragment im Rasterizer

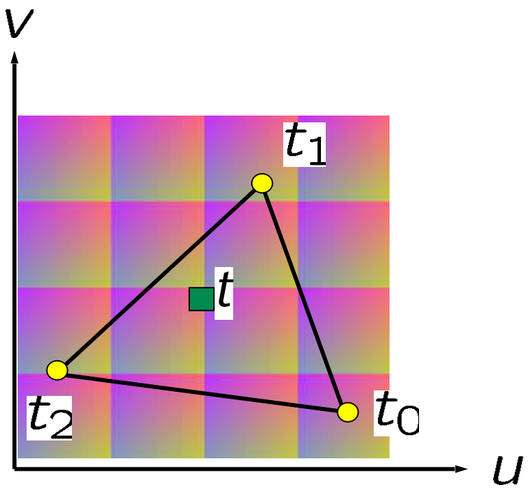
- Erinnerung: baryzentrische Koordinaten

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



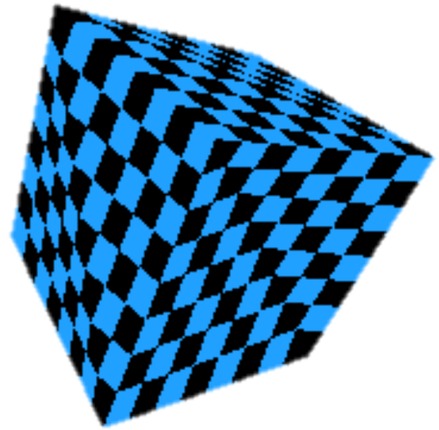
- Textur-Koordinaten durch (simplistische) baryzentrische Interpolation:

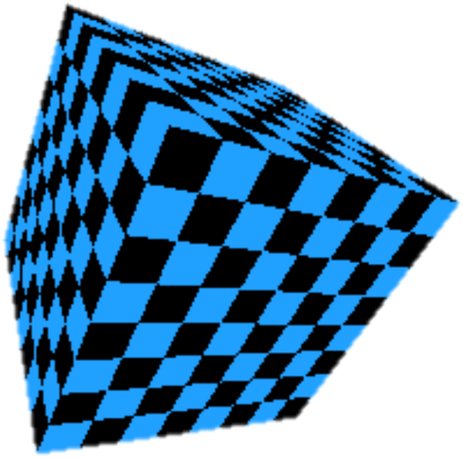
$$t(P) = \sum_{i=0}^{2} \lambda_i(P)\, t_i \qquad t_i \in \mathbb{R}^2$$



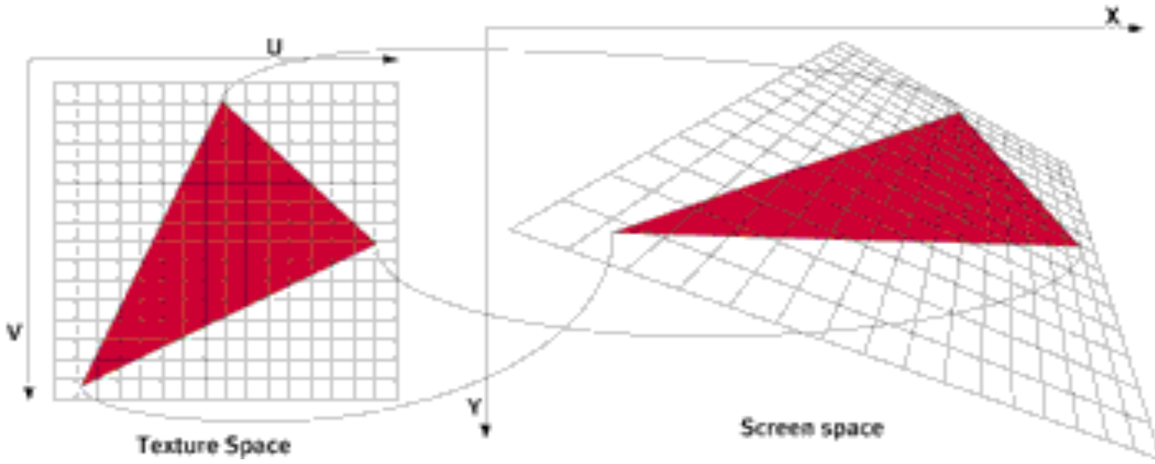
Perspektivisch korrekte Texturkoordinateninterpolation

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





Demo (mehr auf der VL-Homepage)



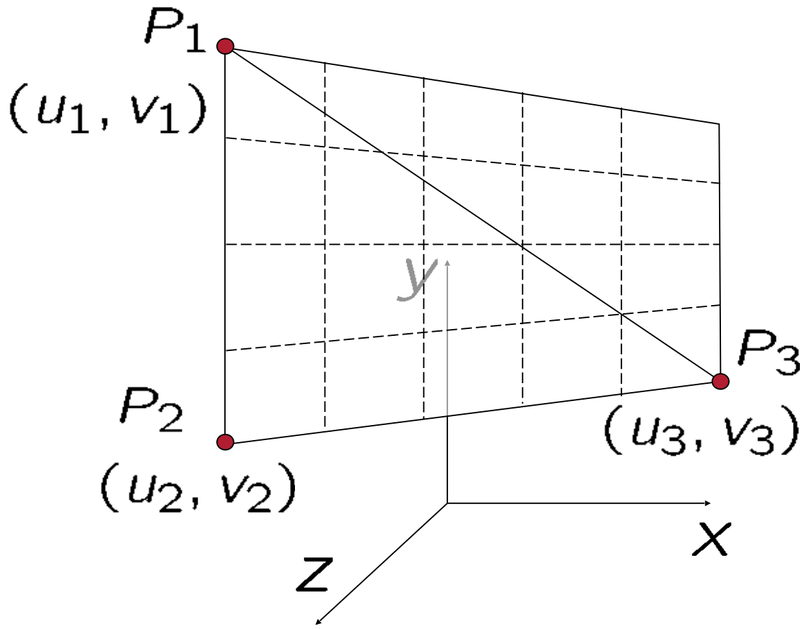
FYI (nicht klausurrelevant)

- Erinnerung: was passiert bei der perspektivischen Projektion

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

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

$$z_0 = \text{Proj.ebene}$$



- Betrachte im folgenden P_1, und P_2

FYI

- Betrachte im Folgenden nur die Interpolation auf einer Linie
  - Für baryzentrische Interpolation geht das Argument analog

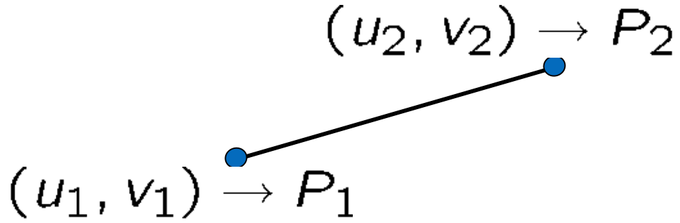
- Gegeben: *t,* zur linearen Interpolation zwischen $\hat{P}_1$ und $\hat{P}_2$, d.h.

$$\hat{P}(t) = t\hat{P}_1 + (1-t)\hat{P}_2$$

- 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 für $\hat{P}$ sind



FYI

- Problem:

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

$$\hat{P}(s) = s\hat{P}_1 + (1-s)\hat{P}_2 \quad s \in [0,1],\ \hat{P}_i = \mathrm{Proj}(P_i)$$

ergeben zwar dieselbe Gerade auf dem Bildschirm, wenn $P(t)$ projiziert wird, aber i.A. ist

$$\mathrm{Proj}(\,P(t)\,) \neq \hat{P}(t)\ !$$

- Frage: wie sieht $\mathrm{Proj}(P(t))$ aus?

FYI

- 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:

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

FYI

- Beweis:

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

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

$$\frac{x_1 w_1 + tw_2 x_1 - tw_1 x_1 + tw_1 x_2 - tw_2 x_1}{w_1^2 + tw_2 w_1 - tw_1^2} =$$

$$\frac{x_1 w_1 - tw_1 x_1 + tw_1 x_2}{w_1^2 + tw_2 w_1 - tw_1^2} = \frac{x_1 - tx_1 + tx_2}{w_1 + tw_2 - tw_1} =$$

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

FYI

- 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 *s* passt zu einem gegebenen *t*, d.h., gesucht ist *s* so daß

$$\text{Proj}(P(t)) = \hat{P}(s)$$

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

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

$$\Rightarrow \quad t = \frac{sw_1}{w_2 + s(w_1 - w_2)}$$

- Mit diesem *t* kann man die Texturkoordinaten *(u,v)* linear interpolieren!

FYI

- Analog funktioniert es bei 3 baryzentrischen Koordinaten:

$$u(P) = \frac{\sum_{i=0}^{2} \lambda_i(P) u_i}{\sum_{i=0}^{2} \lambda_i(P) w_i}$$

Moment Mal!

- War die Interpolation von Farben in Dreiecken falsch?
- Was ist der Unterschied zwischen Interpolation von Farben und der Interpolation von Textur-Koordinaten?!
- Kein Unterschied ...
- Dann hätten wir Farben auch perspektivisch korrekt interpolieren müssen!
- Richtig. Sieht man aber (meistens) nicht ...

Diskrete 3D-Texturen Ab hier wieder klausurrelevant!

- Funktionieren ganz analog zu 2D-Texturen

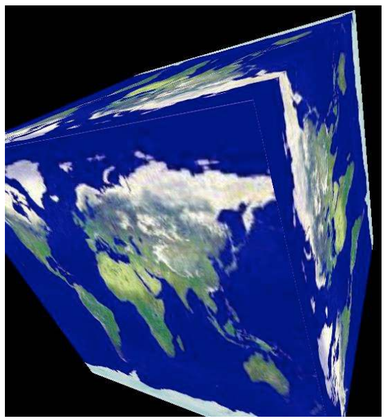
Modulation der Beleuchtung durch Texturen

- Wie kann ein Texturwert (Texel) die Beleuchtungsrechnung beeinflussen? (Was kann man mit einer Textur machen?)
- Erinnerung: das Blinn-Phong-Model

$$L_{\text{out}} = k_d L_a + \left(k_d(\mathbf{n}\cdot l) + k_s(\mathbf{n}\cdot\mathbf{h})^p \right) L_{\text{in}}$$

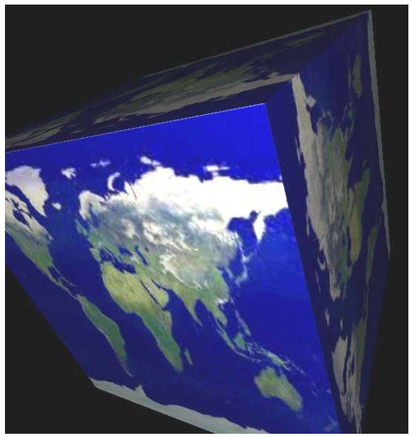
1. Ersetzen der Objektfarbe (*replace*)
   - Einfachste Art der Texturierung
   - Jegliche Beleuchtung wird entfernt

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

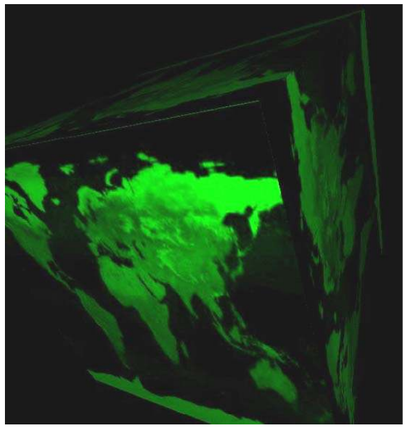


2. A posteriori Skalierung der Farbe (*modulate*)
   - Häufigste Art der Texturierung
   - Komponentenweise Skalierung des Farbwertes

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



Textur auf weißem Würfel



Textur auf grünem Würfel

3. A priori Skalierung der Materialfarbe (Reflexionskoeffizient):

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

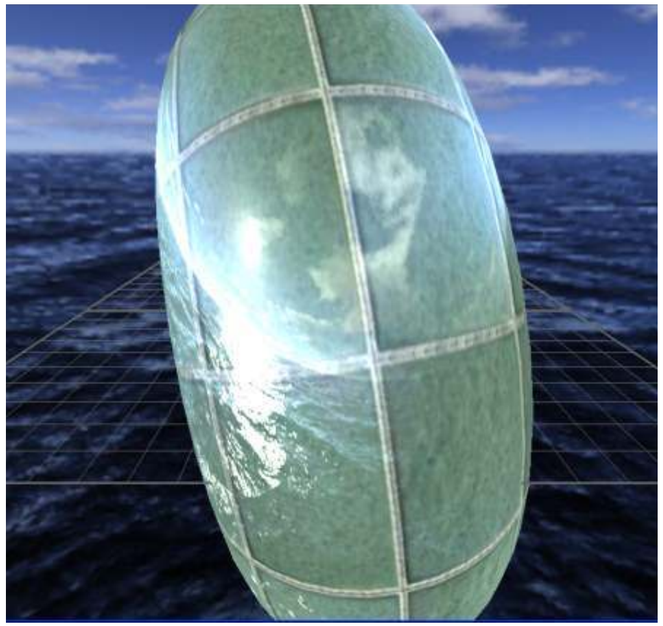
- Textur wird also vor der Berechnung des Beleuchtungsmodells ausgelesen
- Wichtig: im Unterschied zu (2) bleibt hier der spekulare Anteil von der Textur unbeeinflusst

4. Modulation der spekularen Reflexion (*gloss map, specular map*)

- Analog zu (3) für k_s

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

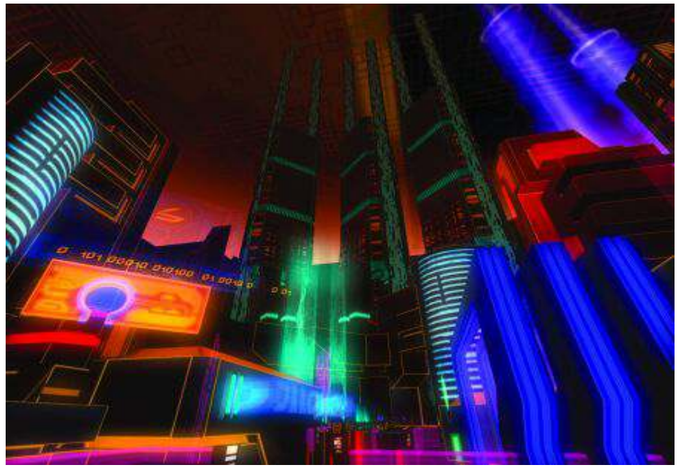
- Erlaubt Modellierung unregelmäßiger *"shininess"* (z.B. verschmutzte / fettige Flächen)



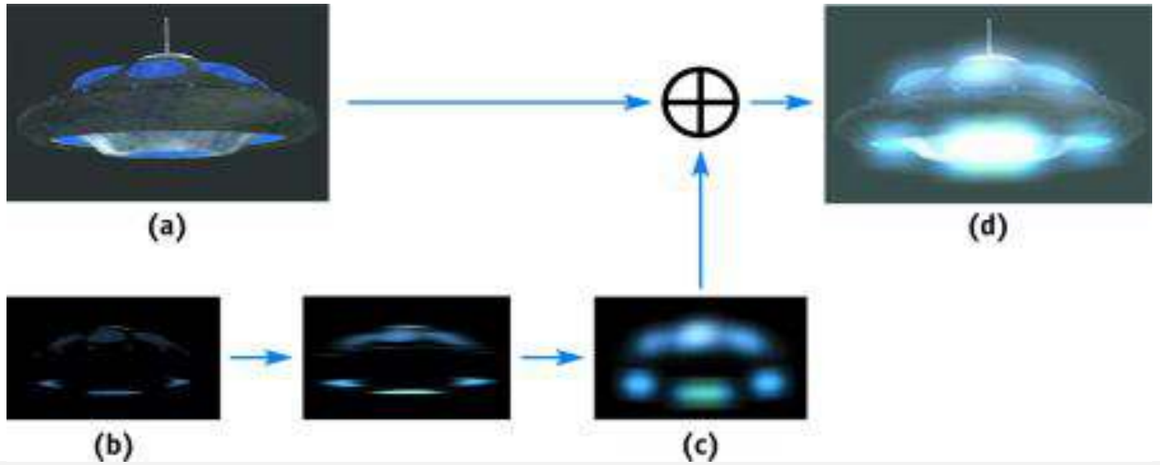
5. "Glow"-Effekt:

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

- For neon signs, TV, laser beams etc.

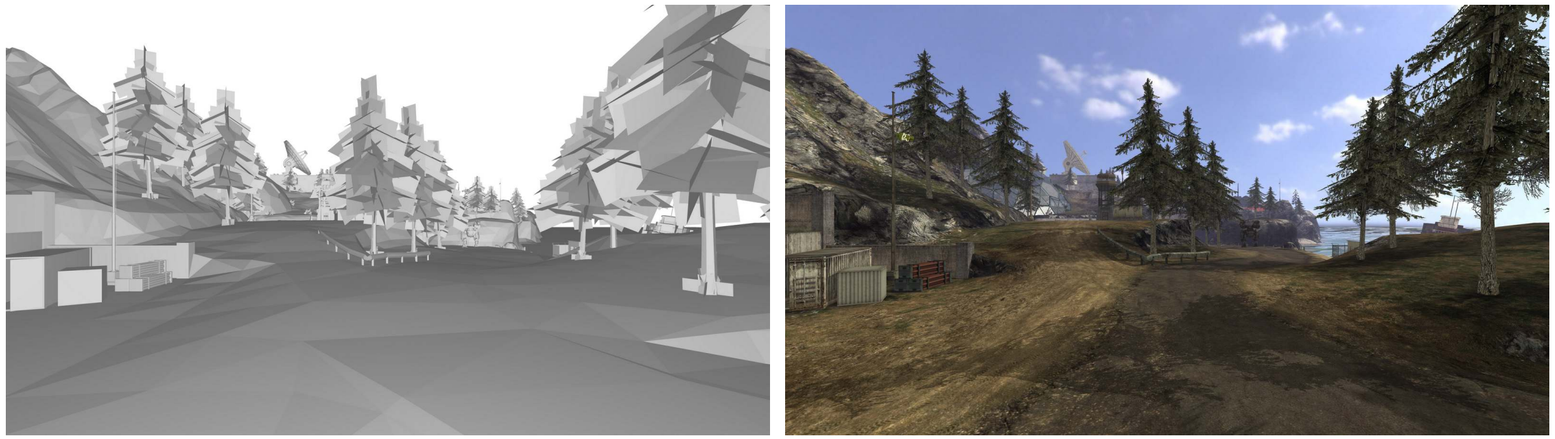


- Solche Effekte (Halos) gehen aber nur mit Multi-Pass-Rendering (Filterung):



6. Modulation der Transparenz

- Speichern der „Durchsichtigkeit“ in einer Textur: $(r, g, b, \alpha)_{x,y} = C_{\text{tex}}(u, v)$
- Pixel mit α=0 sind voll durchsichtig und Pixel mit α=1 sind voll undurchsichtig (opak)
- Ermöglicht komplexe Umrisse



Enemy Territory: Quake Wars. Splash Damage & Intel