13. Texturen

Motivation:
in der Realität Vielzahl von geometrischen Feinstrukturen
Beispiele:

- Maserungen und Muster von Holz, Marmor, Tapeten, Stoffen...
- Wolken, Rauch
- unebene Flächen wie Rauhfaserwände, rauhes Leder, Apfelsinenschalen, Baumstämme
- im Hintergrund sichtbare Häuser, Maschinen, Pflanzen, Personen

Detaillierte Oberflächen erfordern bei polygonaler Modellierung sehr viele, kleine Polygone
⇒ nicht akzeptabel in Bezug auf Rendering-Zeit und Speicherplatz
⇒ andere Möglichkeiten der Darstellung von Details notwendig

pragmatische Lösungen gefragt:

"All it takes is for the rendered image to look right."
(Jim Blinn)

Texturen:
die Oberfläche z.B. einer Wand kann (zunächst) als ein einziges Polygon modelliert werden. Ein "Tapezieren" kann als Aufbringen eines Bildes auf die Oberfläche interpretiert werden. Diesen Vorgang bezeichnet man als "*Texturierung*".

3 Aspekte:

- Hinzufügen eines Musters zu einer glatten Fläche: *texture mapping*.
- Hinzufügen des Eindrucks von Rauheit zu einer glatten Fläche: *displacement mapping* und *bump mapping*
- Simulation der Umgebung auf der Fläche: *environment mapping*

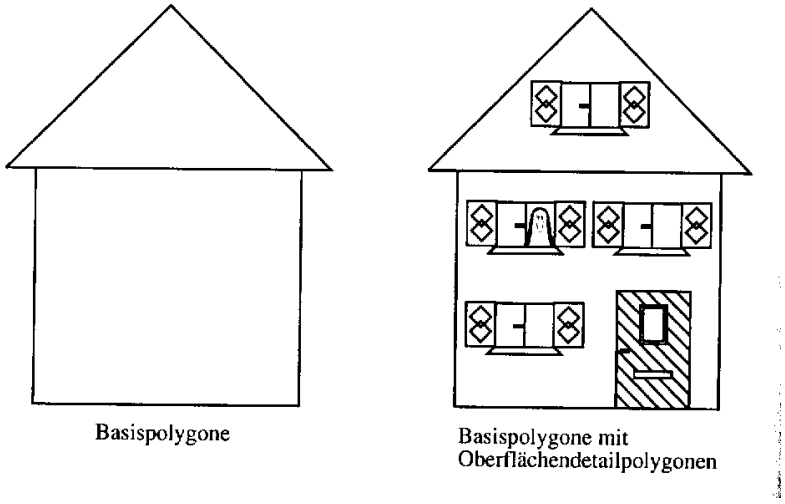
Geschichte von Texturen

- **Catmull/Williams`74: Grundlegende Idee**
- **Blinn und Newell`76: Reflection-Maps**
- **Blinn`78: Bump-Mapping**
- **Williams`78, Reeves`87: Shadow-Maps**
- **Smith`80, Heckbert`83: Texturierung von Polygonen**
- **Williams`83: MipMaps**
- **Miller/Hoffman`84: Illumination/Reflectance-Maps**
- **Perlin`85/Peachey`85: Solid Textures**
- **Green`86: Environment-Maps**
- **Akeley`93: Reality-Engine mit Textur-Hardware**

Darstellung im Folgenden nach Schlechtweg 2001, Jackèl 2001, Krömker 2001, Slusallek 2000, Encarnação et al. Bd. 2, 1997; Bungartz et al. 1996

Einfachster Ansatz:
Oberflächendetailpolygone, die auf die Basispolygone koplanar aufgebracht werden

- beim Sichtbarkeitsentscheid oder Raytracing werden zunächst nur die Basispolygone getestet
- ist ein solches ganz unsichtbar, so gilt dasselbe auch für alle Detailpolygone
- bei der Beleuchtungsrechnung sind die Oberflächendetail-polygone maßgeblich



Basispolygone

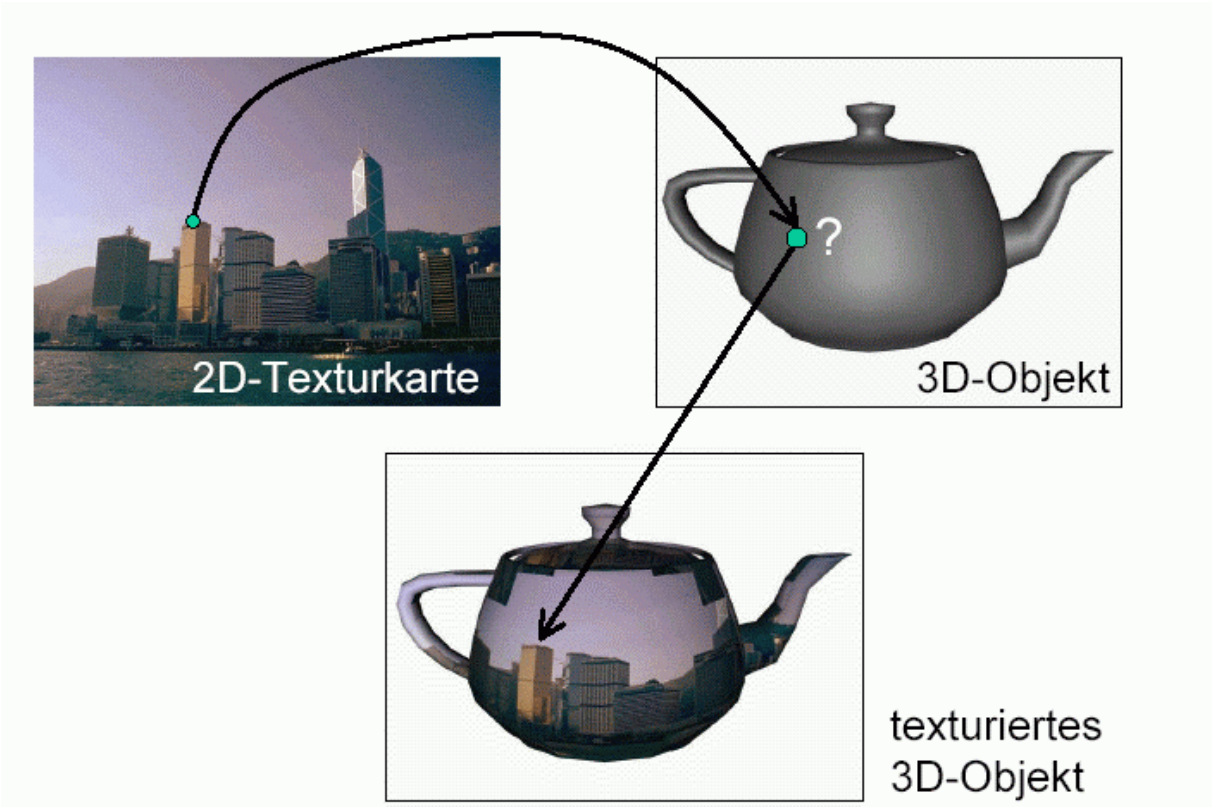
Basispolygone mit Oberflächendetailpolygonen

Der Ansatz ist bei vielen Details noch zu aufwändig (zu viele Detailpolygone)

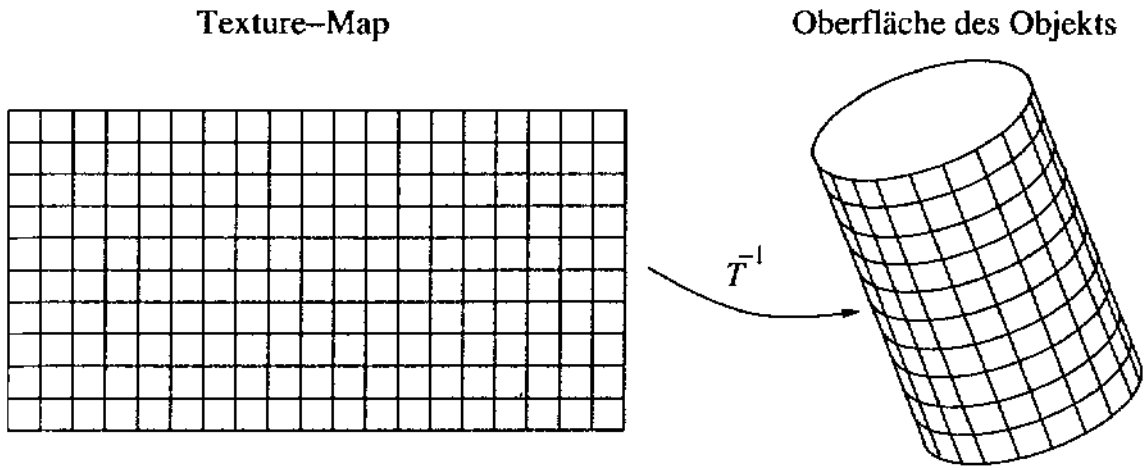
Alternative:

Texture Mapping

Problemstellung:
Gegeben ist eine Textur in Form eines 2D-Bildes, aufgelöst in Textur-Pixel, die man auch "Texel" (Textur-Elemente) nennt. Wie werden diese Elemente auf die Oberfläche eines darzustellenden 3D-Objektes abgebildet?



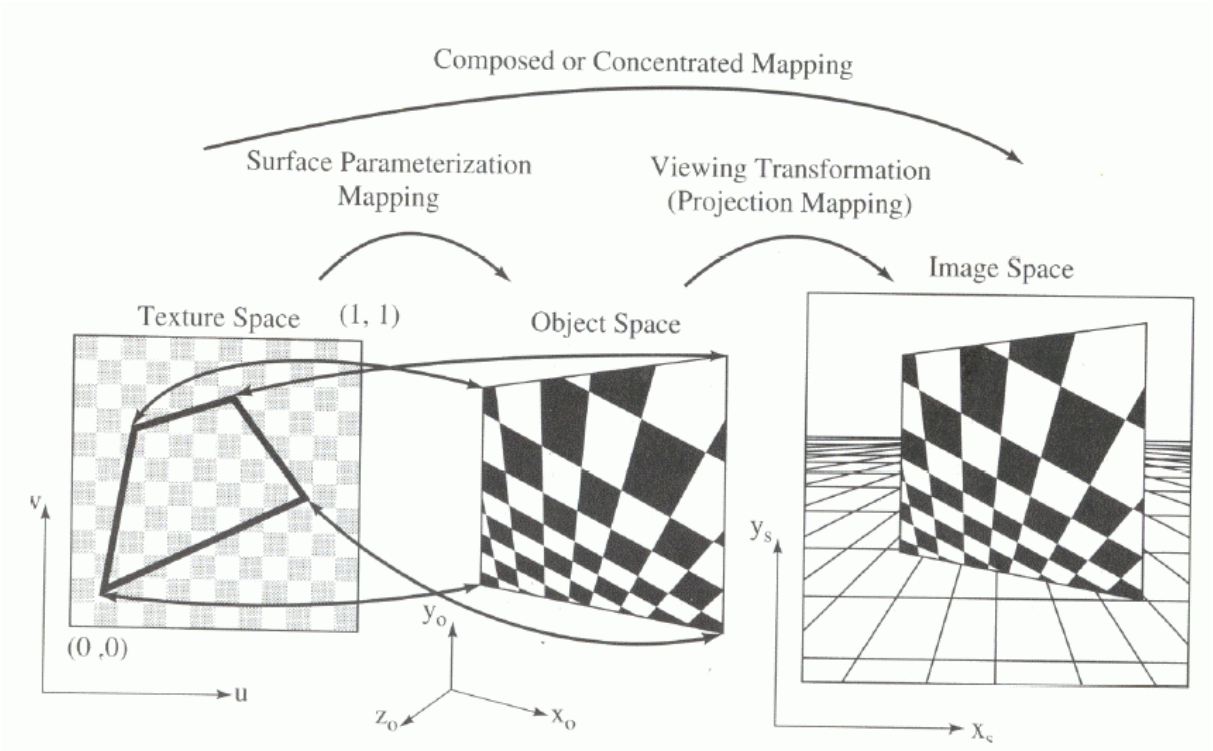
Das *Mapping* beschreibt, wie eine 2D-Textur bzw. ein Ausschnitt davon auf eine Fläche aufgebracht wird.
Beim Rendering muss jedoch das *inverse Mapping-Problem* gelöst werden, d.h. den bekannten (*x*, *y*, *z*)-Koordinaten eines Punktes der Oberfläche im Objektraum müssen Texel-Koordinaten (*u*, *v*) (oft auch (*s*, *t*), siehe VRML) zugeordnet werden.

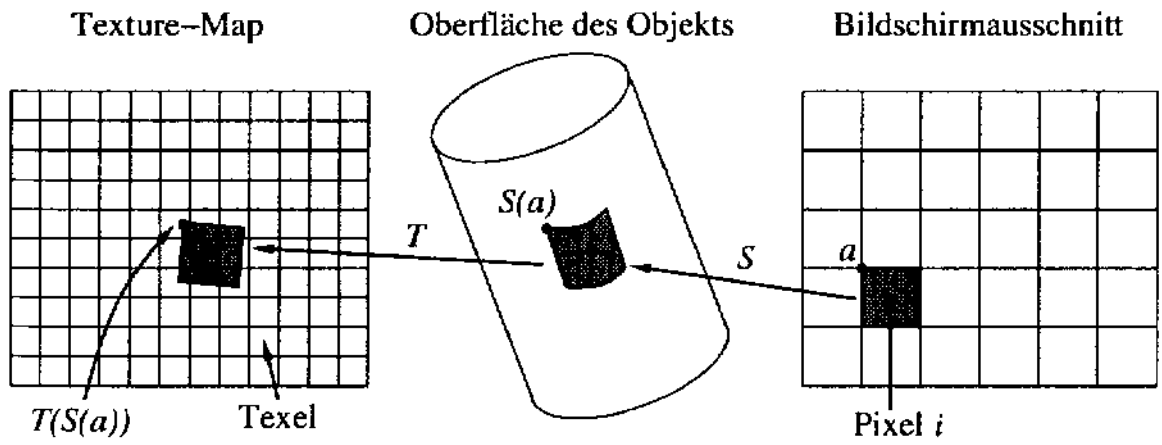


Insgesamt werden 2 Transformationen (Mappings) benötigt:

- Texturkoordinaten in Objektkoordinaten
- Objektkoordinaten in Bildkoordinaten (*viewing transformation*)

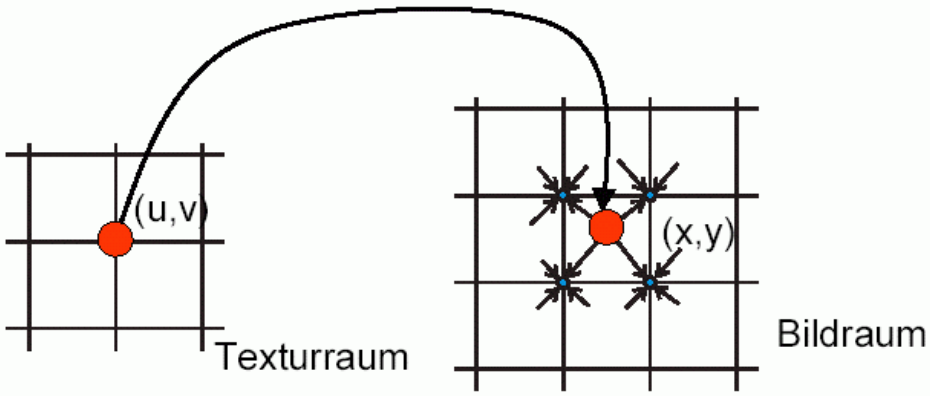
häufig werden beide Transformationen zu einer zusammengefasst (*composed mapping*, Texturabbildung vom Texturraum in den Bildraum).





Zusammengesetzte, inverse Abbildung:

Texturabbildung vom Texturraum in den Bildraum:



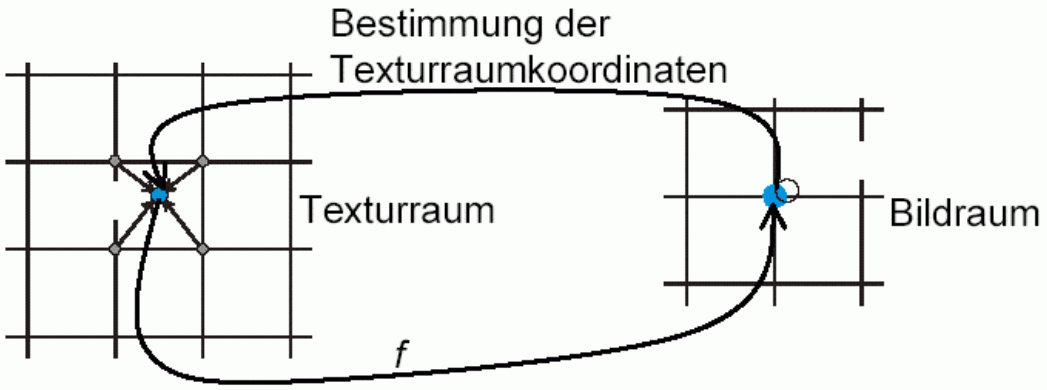
Der Farbwert des transformierten Pixels wird auf die benachbarten, ganzzahligen Bildraumkoordinaten aufgeteilt.

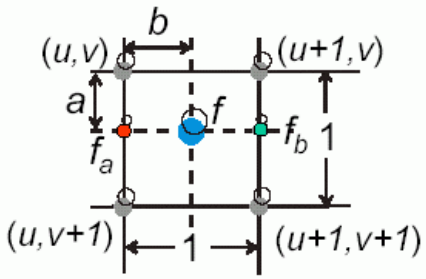
Einfacher Ansatz:
den Objektraum (und dortige, kompliziertere Strukturen) "umgehen" oder vereinfachen und die Texturabbildung (bzw. ihre Inverse) direkt mit einfachen Funktionen beschreiben

Varianten:

- affine Texturabbildung
- perspektivische Texturabbildung
- bilineare Texturabbildung
- "Two-Part" Texturabbildung mit einfacher Hilfsfläche anstelle der Original-Oberfläche im Objektraum

Inverse Texturabbildung :





Bilineare Interpolation:

$$f_a = a\,f(u,v) + (1-a)\,f(u,v+1)$$

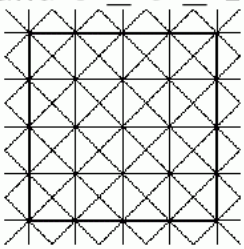
$$f_b = a\,f(u+1,v) + (1-a)\,f(u+1,v+1)$$

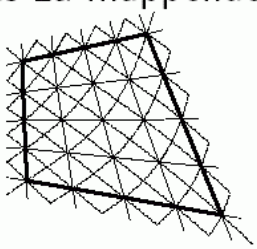
$$f = b\,f_a + (1-b)\,f_b$$

- Bilineares Mapping
  - verwendet u. a. bei Gouraud- und Phong-Shading
  - transformiert ein Quadrat in ein allgemeines Viereck
  - allgemeine Form:

$$q(u,v) = (1-u)(1-v)P_{00} + u(1-v)P_{10} + (1-u)vP_{01} + P_{11}$$

  - $P_{ij} \quad (i = 0,1;\ j = 0,1)$ Eckpunkte des Vierecks im Ziel-Koordinatensystem
  - $0 \leq u \leq 1$ und $0 \leq v \leq 1$ definiert das zu mappende Quadrat

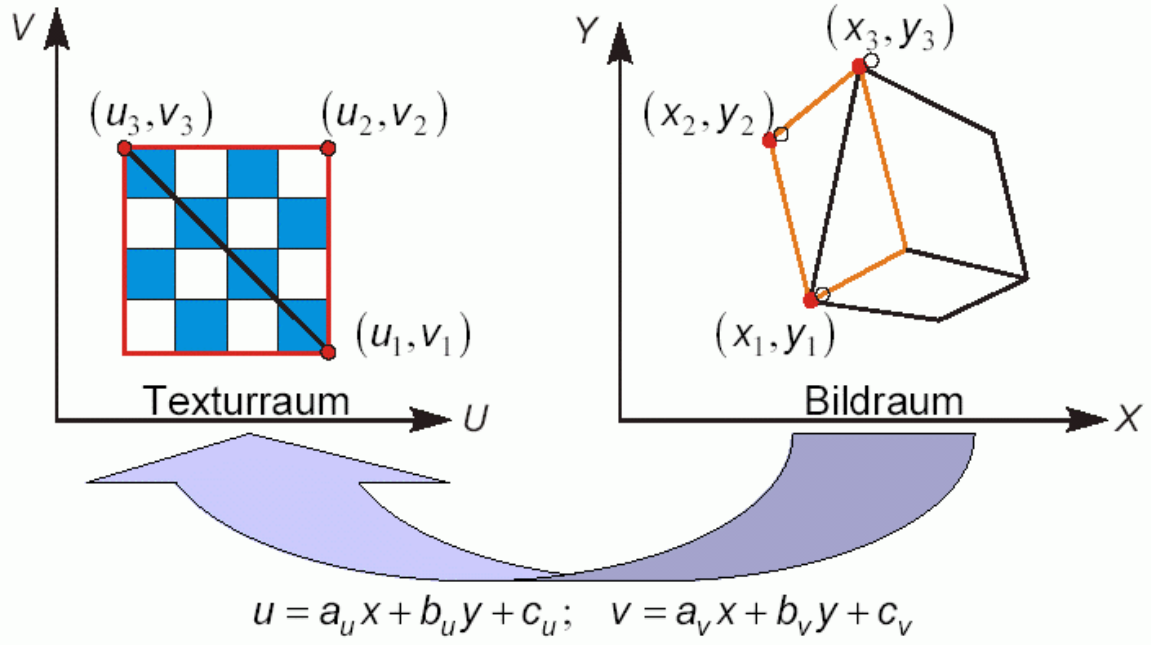




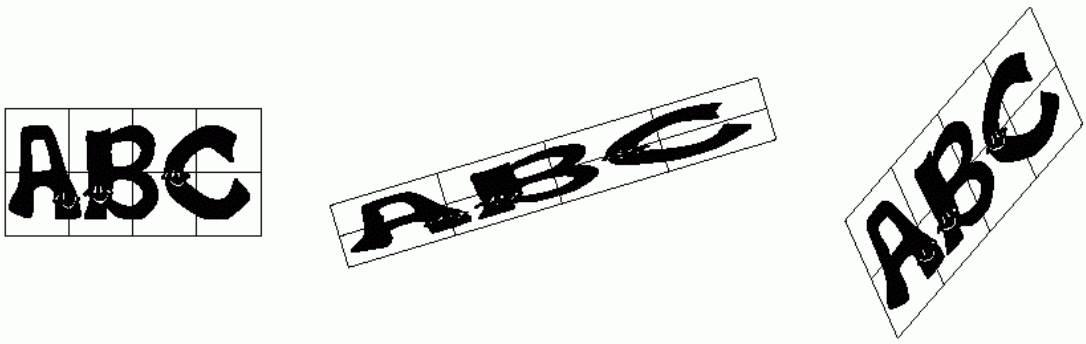
- vertikale und horizontale Geraden bleiben Geraden
- diagonale Geraden werden zu Kurven
- Kompositionen bilinearer Transformationen sind biquadratisch, schwer handhabbar

Affine Texturabbildung:

Bestimmung der Texturraumkoordinaten (u,v) als Funktion der Bildraumkoordinaten (x, y)

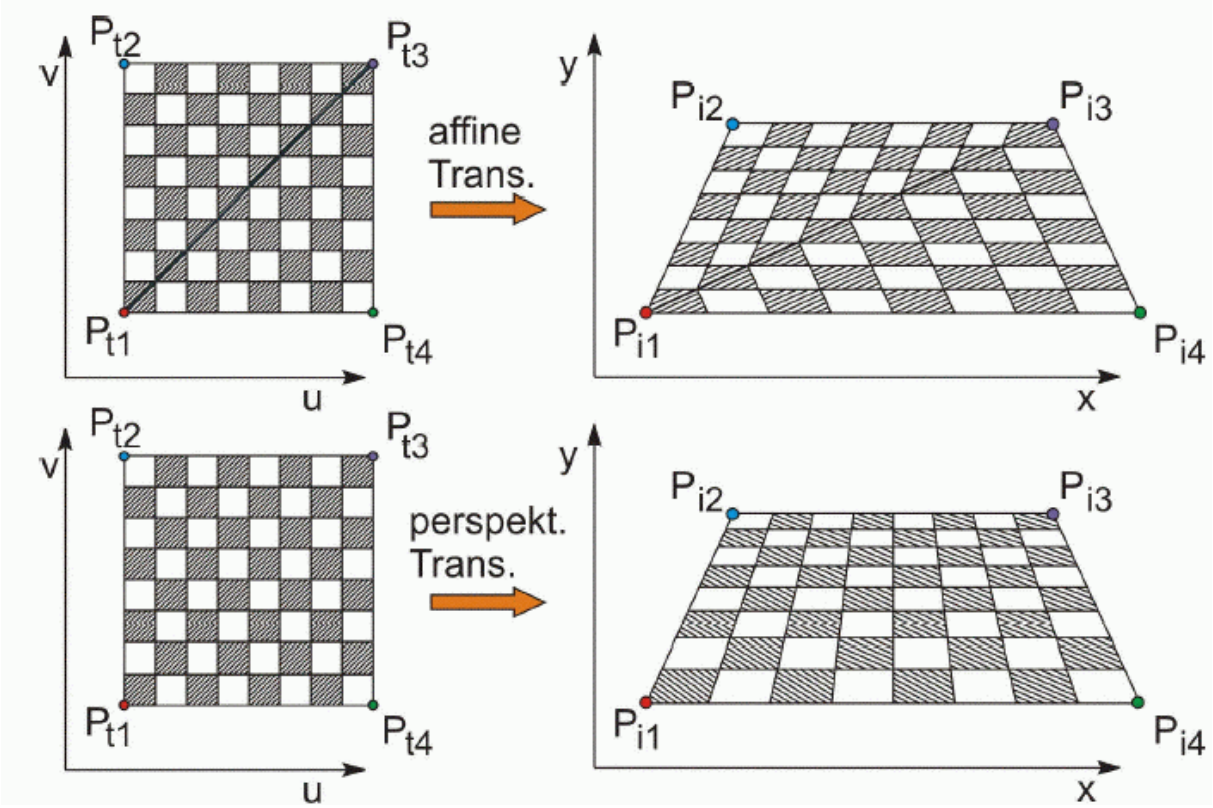


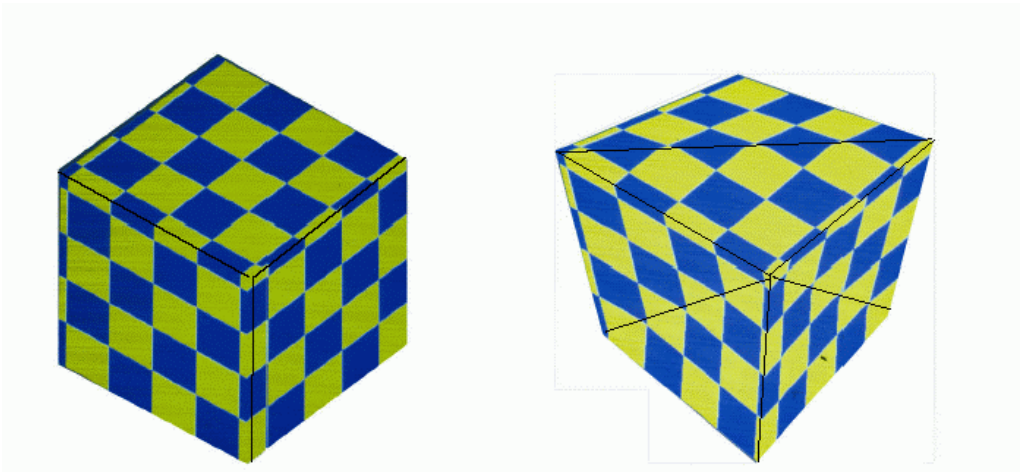
- Affine Mapping-Transformation
  - Skalierungen, Rotationen, Translationen, Scherungen und Kombinationen davon
  - erhalten parallele Linien
  - Dreiecke → Dreiecke, Rechtecke → Parallelogramme



Problem der affinen Texturabbildung:
bei Anwendung auf perspektivisch projizierte Bilder führt die affine Interpolation aus den Texturwerten an den Eckpunkten zu Artefakten ("Bruchlinien").

Vergleich affine Texturabbildung vs. perspektivische Texturabbildung:

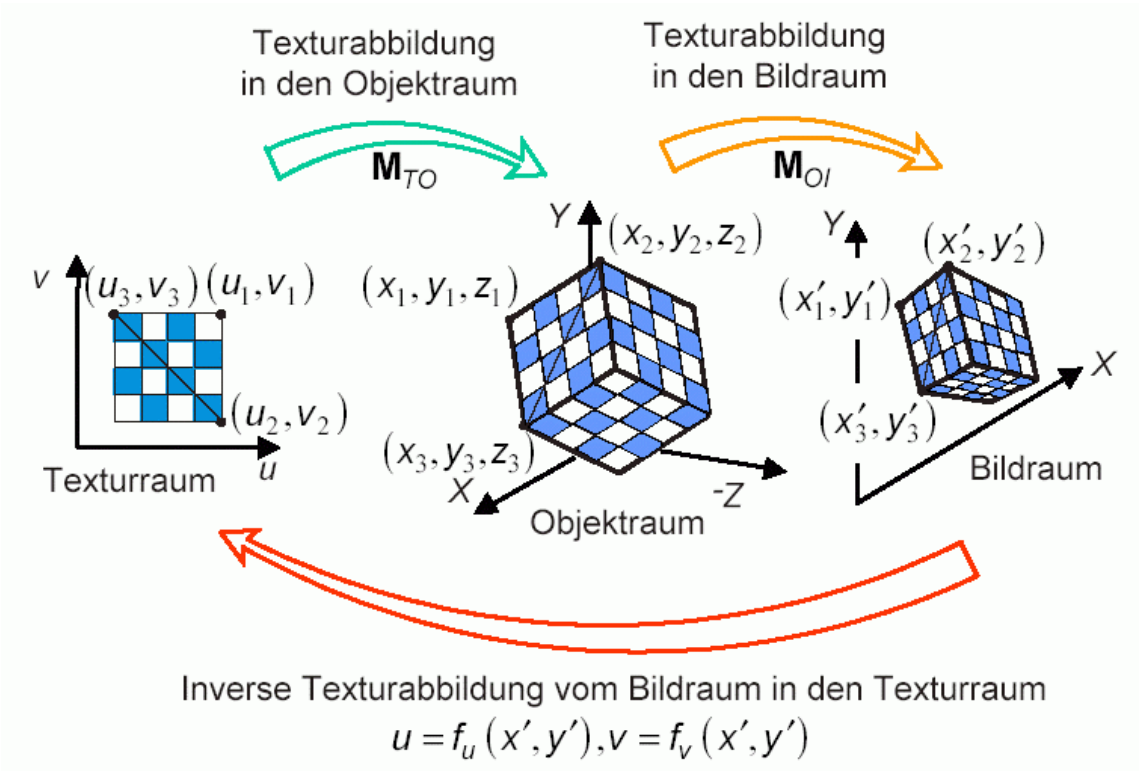




Affine Texturabbildung auf parallel projiziertes Objekt

Affine Texturabbildung auf perspektivisch projiziertes Objekt

Die *perspektivische Texturabbildung* ("projektives Mapping") berücksichtigt die perspektivische Verzerrung beim Übergang vom Objektraum in den Bildraum. Rechnerisch ergibt sich eine gebrochenrationale Funktion:



Inverse Texturabbildung vom Bildraum in den Texturraum

$$u = f_u(x', y'), v = f_v(x', y')$$

Bestimmung von: $u = f_u(x', y'), v = f_v(x', y')$

$$[u \quad v \quad 1] \overset{\mathbf{M}_{TO}}{\Rightarrow} [x \quad y \quad z \quad 1] \overset{\mathbf{M}_{OI}}{\Rightarrow} [x' \quad y' \quad 1]$$

Texturraum — Objektraum — Bildraum

Transformationsmatrix:
Texturraum nach Objektraum

$$\mathbf{M}_{TO} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ 0 & 0 & 1 \end{bmatrix}$$

Transformationsmatrix:
Objektraum nach Bildraum

$$\mathbf{M}_{OI} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

Zentral-projektion

Transformationsmatrix: Texturraum nach Bildraum

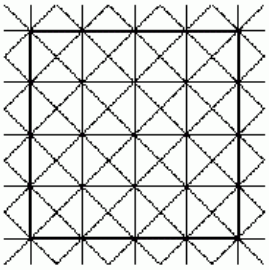
$$\mathbf{M}_{TI} = \mathbf{M}_{OI} \times \mathbf{M}_{TO} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31}/d & a_{32}/d & a_{33}/d \end{bmatrix}$$

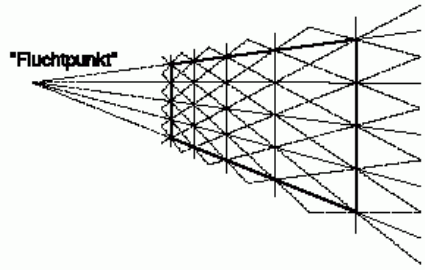
Man erhält:
u von der Form $u = (ax' + by' + c) / q$,
v von der Form $v = (dx' + ey' + f) / q$,
mit $q = gx' + hy' + j$.

- projektives Mapping
  - „virtueller“ Fluchtpunkt, in dem sich die Linien treffen
  - Berechnung Texturkoordinaten → Bildkoordinaten:

$$x_s = \frac{au + bv + c}{gu + hv + i} \qquad y_s = \frac{du + ev + f}{gu + hv + i}$$

  - Inverse hat eine ähnliche Form mit anderen Koeffizienten
  - Ableitung der Koeffizienten für jedes texturierte Polygon
  - erhält Linien, aber generell keine gleichen Abstände





- Bilineares Mapping wird manchmal (eher selten) verwendet.
- Affines und projektives Mapping wird bevorzugt: inverse Abb. kann einfacher berechnet werden

Beachte: affines Mapping von Texturkoordinaten zu Objektkoordinaten und perspektivische Projektion beim Rendering resultiert in projektivem Mapping von Texturkoordinaten in Bildkoordinaten.

Mehrere *Algorithmen* zum eigentlichen Texturieren:

- häufig angewandt: Bildraum-Scan

```
foreach y' do
   foreach x' do
      { berechne u(x', y'), v(x', y')
        kopiere texture(u, v) nach bild(x', y') }
```

- Kopieren der Textur kann auch durch Manipulation der Beleuchtungsgleichung ersetzt werden.

Die bisher beschriebenen Ansätze betreffen Textur-Mapping für ein einzelnes Polygon.

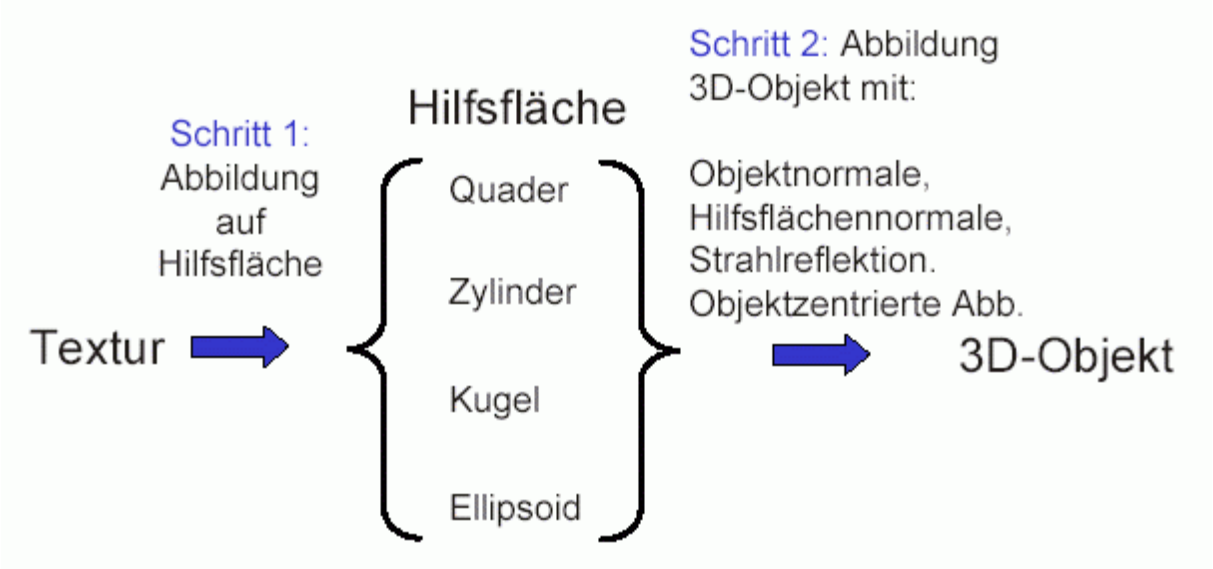
- Besteht ein Objekt aus sehr wenigen Polygonen, so kann ein Anwender die Zuordnung der Polygone zur Textur-Referenzfläche u.U. noch interaktiv vornehmen
- bei komplexeren Objekten ist dies unmöglich

⇒ *Zweischrittverfahren* (*Two-Part Mapping, Two-Pass-Method*)

Automatisierung der (*u*, *v*)-Berechnung, erstmals vorgestellt von Bier & Sloan 1986.

Grundidee: ein komplexes Objekt wird von einer einfach parametrisierbaren, virtuellen Fläche umhüllt (meist Zylinder, Kugel oder Quader),
die 2D-Textur wird dann zunächst einmal auf diese umhüllende Fläche abgebildet, und erst von dort auf die Objektoberfläche.

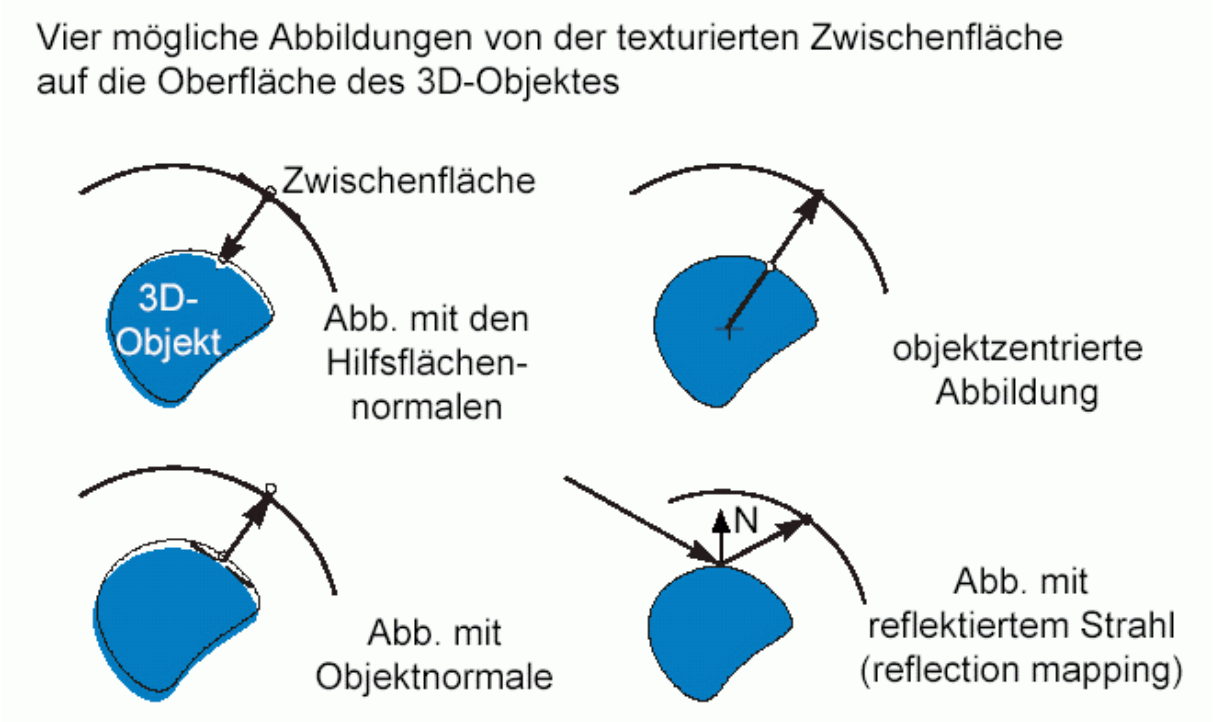
$$\begin{array}{ccccc} & S & & O & \\ (u,v) & \longrightarrow & (x_i, y_i) & \longrightarrow & (x_o, y_o, z_o) \\ \text{texture space} & \longrightarrow & \text{intermediate space} & \longrightarrow & \text{object space} \end{array}$$



einfachster Fall:
planares Mapping
Abbildung auf eine Ebene und dann auf das Objekt

- einfachste Art des Mapping
- allg.: 3 Rotationen, 3 Translationen und Skalierung notwendig, um die Textur in der Ebene zu plazieren
- der wesentliche, ggf. vom Benutzer zu steuernde Teil ist die Skalierung (und eventuelle Wiederholung des Texturmusters auf der Ebene, wenn dies gewünscht ist).

Bei komplizierteren Zwischenflächen:



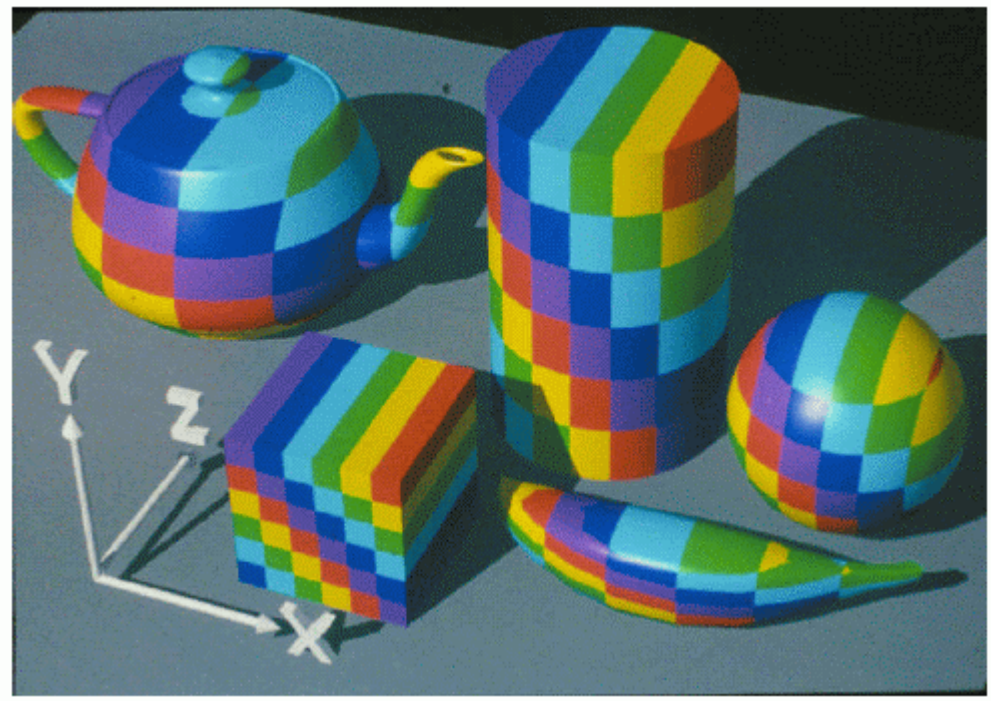
man unterscheidet nach Art der Zwischenfläche:

- Box-Mapping
- Zylinder-Mapping
- Kugel-Mapping

Box-Mapping:

umhüllende Fläche = Quader, i. d. Regel die achsenparallele Bounding Box des Objekts
mögliche Parametrisierung von Punkten in der Box:
längste Kante des Quaders = *u*-Achse, zweitlängste Kante = *v*-Achse

Beispiel für Anwendung:



Zylinder-Mapping:

- Abbildung auf einen Zylinder und dann auf das Objekt
- insbes. nützlich, wenn das Objekt rotationssymmetrisch ist

Mapping-Funktion:

$$(\theta, h) \overset{S}{\longleftarrow} (u, v) = [ar(\theta - \theta_0), d(h - h_0)] \qquad -\pi < 0 < \pi$$

- Diskontinuität ("Naht") an einer Stelle parallel zur Mittelachse

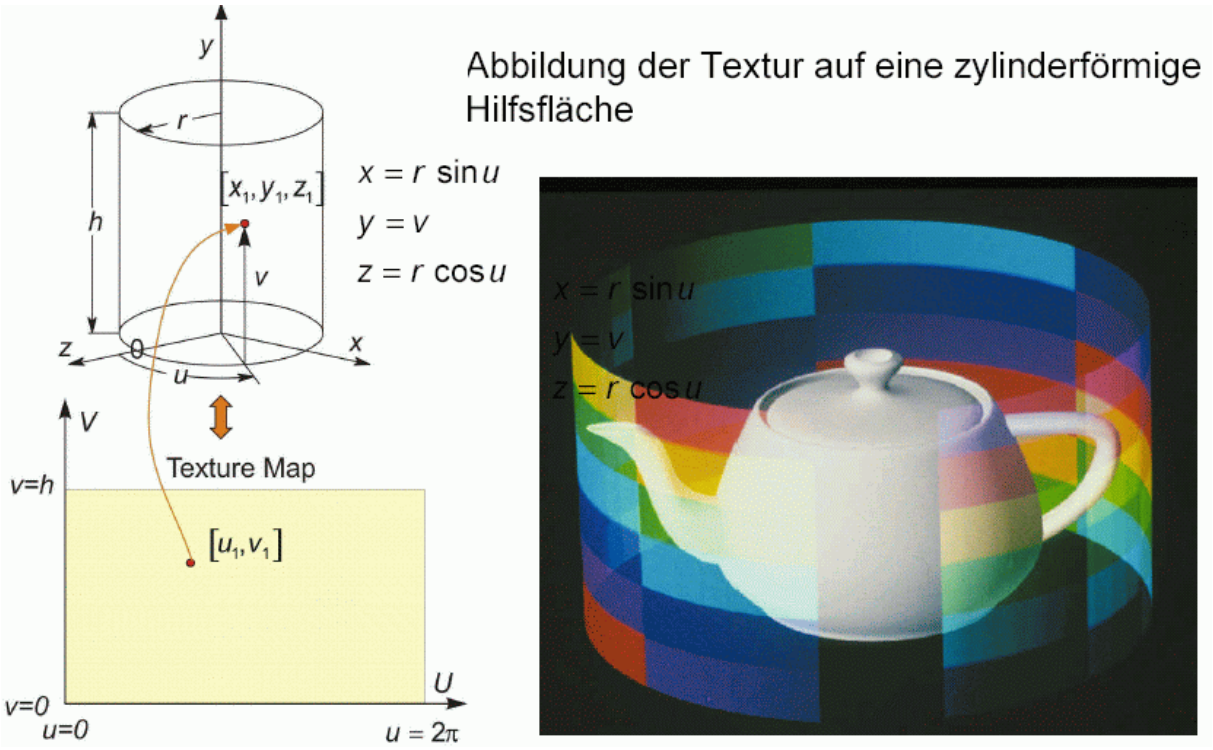
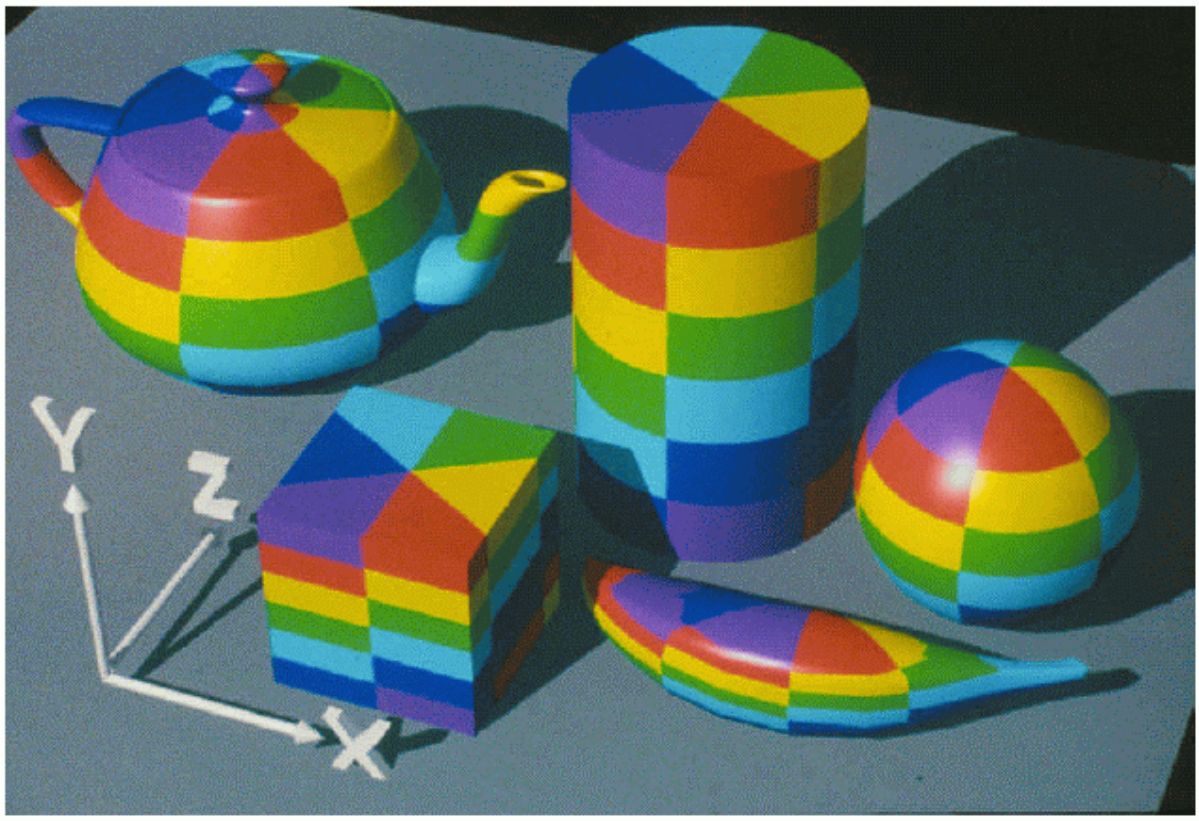


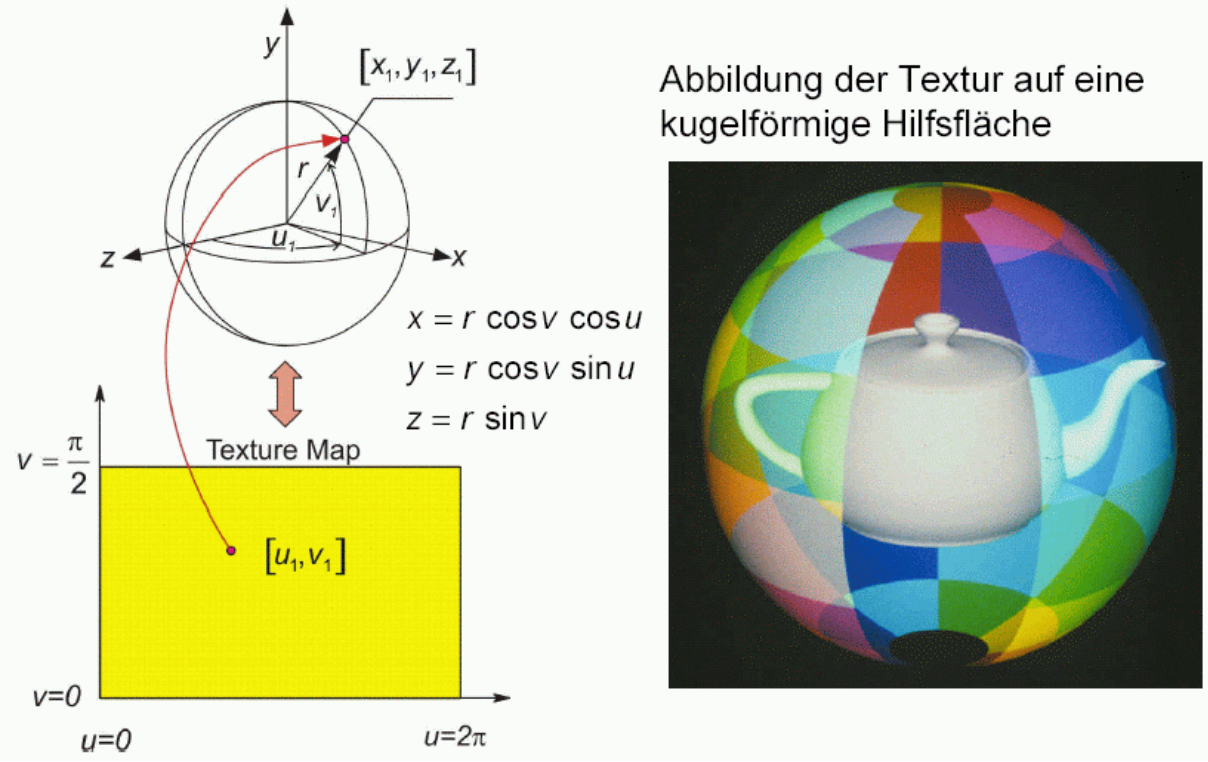
Abbildung der Textur auf eine zylinderförmige Hilfsfläche
y
r
h
$[x_1, y_1, z_1]$
v
z
0
x
u
$x = r \sin u$
$y = v$
$z = r \cos u$
V
Texture Map
v=h
$[u_1, v_1]$
U
v=0
u=0
$u = 2\pi$
$x = r \sin u$
$y = v$
$z = r \cos u$

Anwendung:

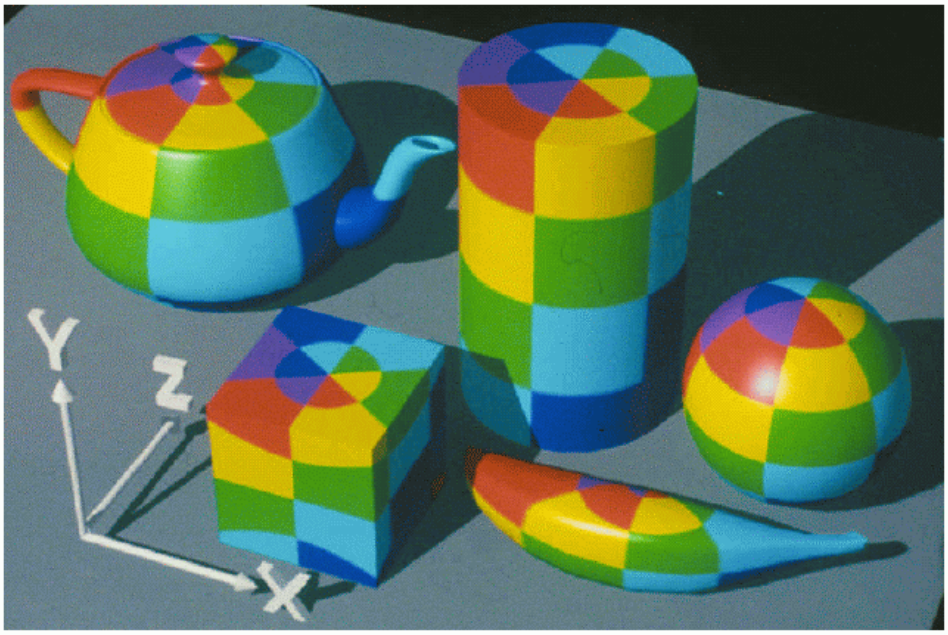

Y
Z
X

Kugel-Mapping (*sphärisches Mapping*):

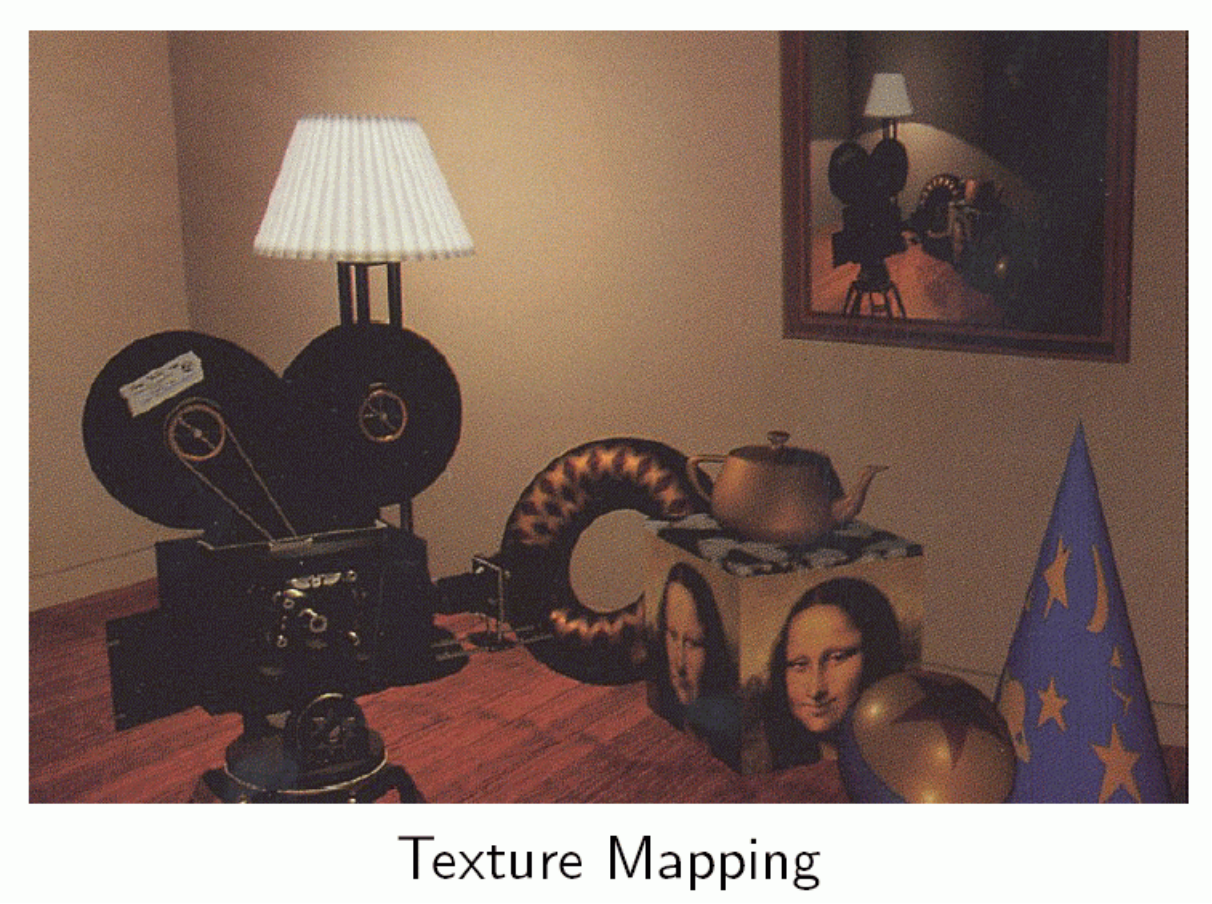
- Abbildung auf eine Kugel (bzw. 2 Halbkugeln) und dann auf das Objekt
- stereografische Projektion für die Abb. auf die Halbkugeln
- Diskontinuität entlang des Äquators
- generell keine verzerrungsfreie Abb. einer ebenen Fläche auf eine Kugeloberfläche und umgekehrt möglich



Anwendungsbeispiel:



Anwendung von Texture Mapping auf das Referenzbild aus dem Kapitel über Beleuchtungsrechnung:



Texture Mapping

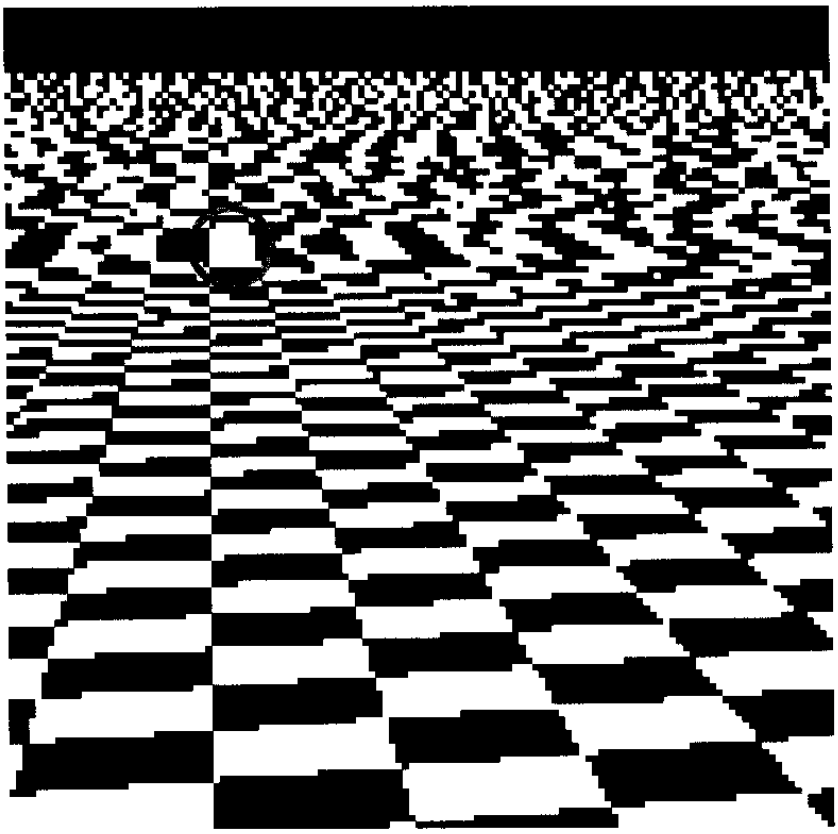
Probleme bei Verwendung von Rasterbildern als Texturen im Texture Mapping:

Aliasing, d.h. visuelle Artefakte, die dadurch entstehen, dass viele Texel auf einen Pixel abgebildet werden können

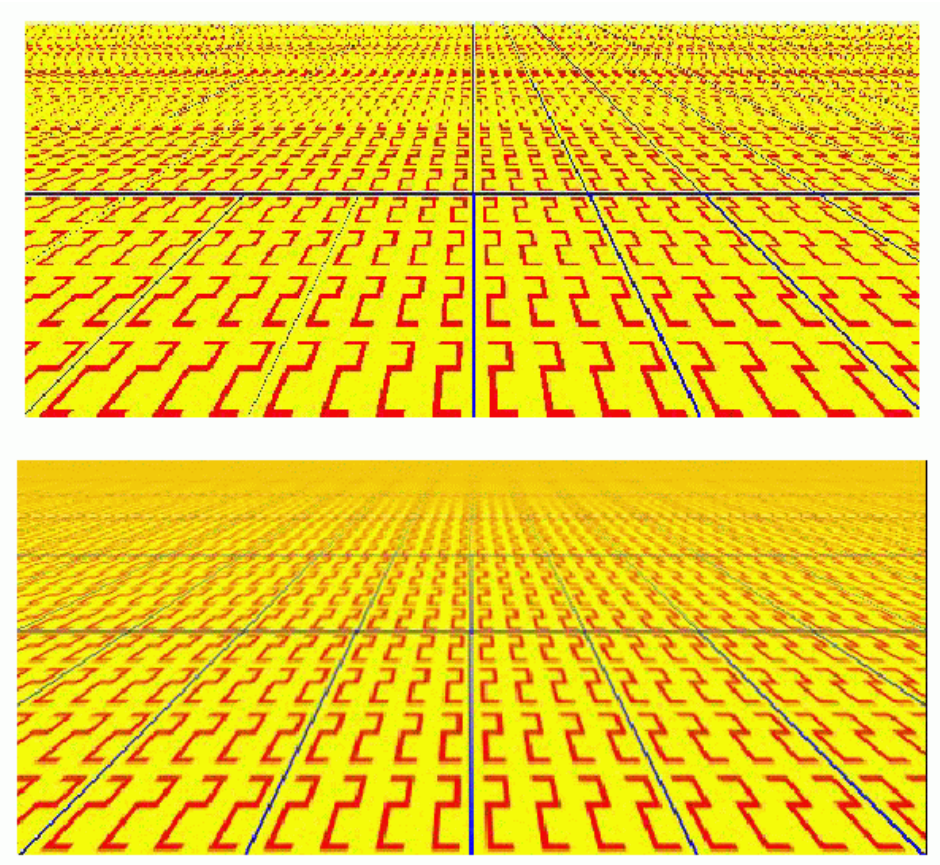
Abhilfe:

- Texturberechnung mit Antialiasing-Techniken (s. Kapitel 5)
- Kontrolle des "Footprints" = Urbild eines Display-Pixels in der Texturebene
- ggf. mit verschiedenen Texturen für verschiedene Auflösungsschärfen arbeiten ("Mip-Mapping", s.u.)

Aliasing-Effekte:



texturierte Ebene ohne und mit Antialiasing:



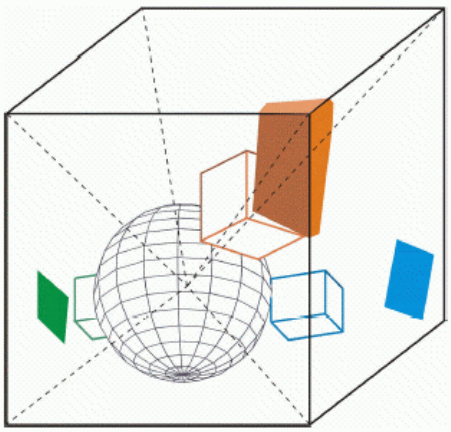
Environment Mapping / Reflection Mapping

= eine Möglichkeit, Reflektionen (approximativ) mit Hilfe von Textur-Hardware (und ohne Raytracing, d.h. schnell) zu berechnen

Grundidee:
Ist ein Objekt klein i. Vgl. zum Abstand zu umgebenden Objekten, so hängt die einfallende Beleuchtungsstärke nur von der Richtung, nicht von der Position eines Punktes auf dem Objekt ab ⇒ die einfallende Beleuchtung kann für ein Objekt vorberechnet und in einer 2D-Textur, der *Environment Map*, gespeichert werden.

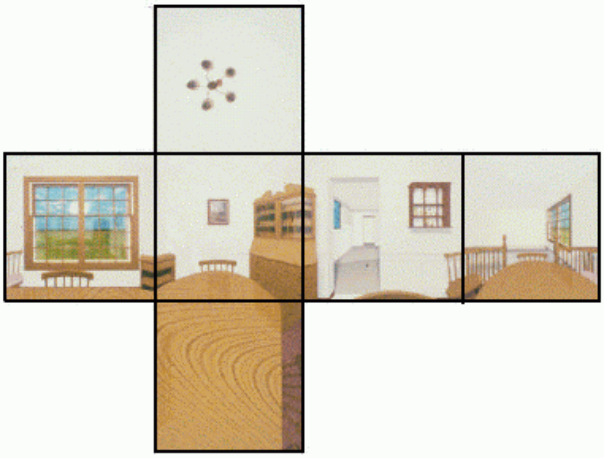
Realisierung:
Das reflektierende Objekt wird von einer virtuellen Zwischenfläche (Kugel, Box...) (wie beim two-pass texture mapping) umgeben, auf deren Innenseite die Szenenumgebung als 2D-Textur (Environment Map) aufgetragen ist.
Einem Punkt *P* auf der Objektoberfläche werden dann Texturkoordinaten (*u*, *v*) über den reflektierten Strahl vom Betrachter zugeordnet.
Für die Beleuchtungsrechnung wird dann ein erweitertes Phong-Modell mit einem Reflektions-Summanden benutzt.

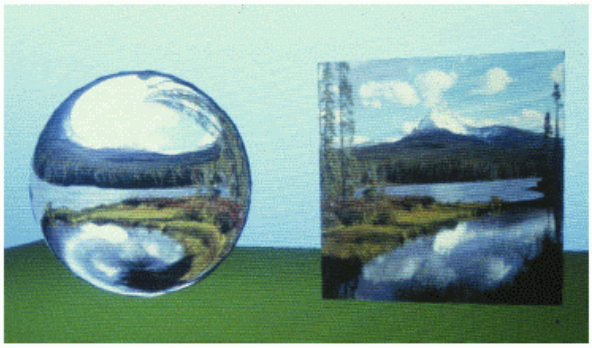
1. Schritt: Projektion der Szene vom Zentrum des zu texturiernden 3D-Objektes auf die sechs Seitenflächen des umgebenden Kubus



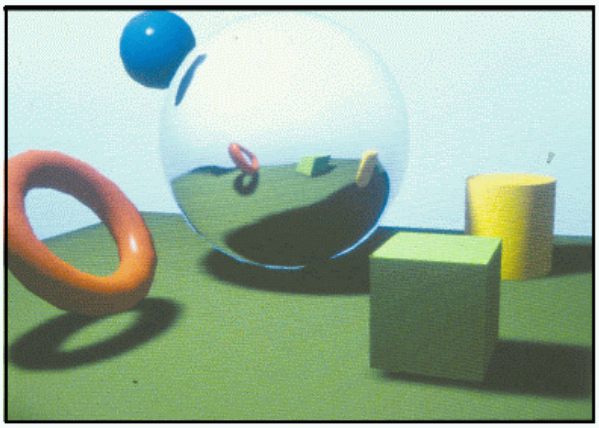
2. Schritt: Reflection Mapping

Beispiel:

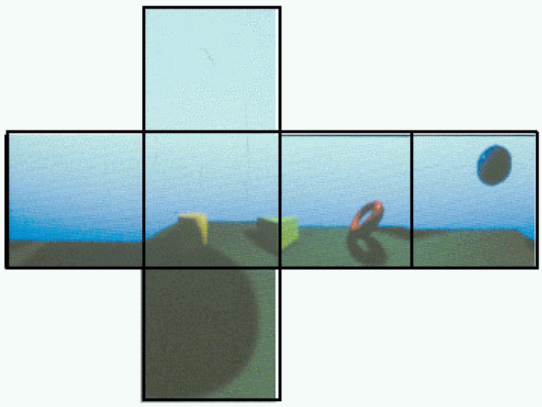




projizierte 3D-Szene



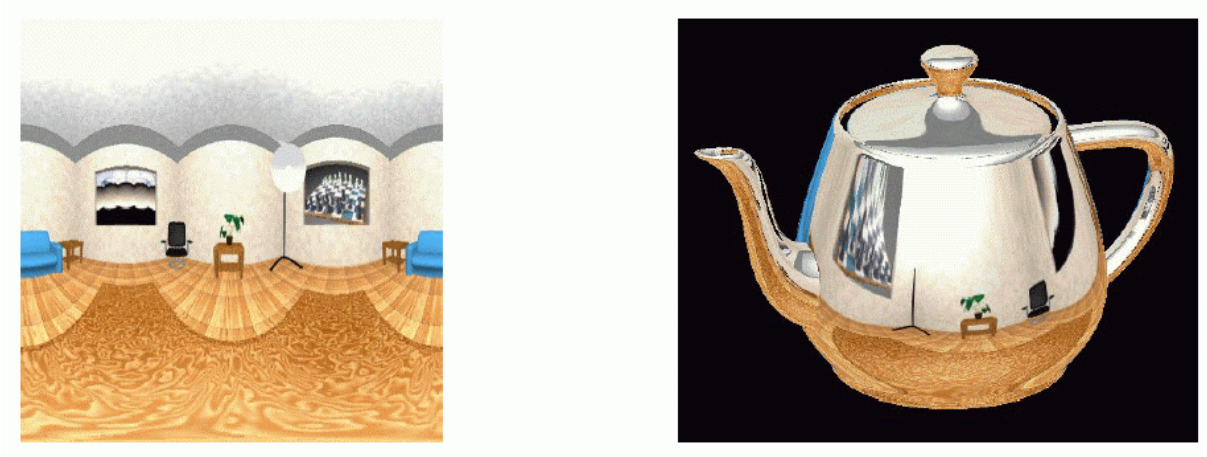
Abwicklung des Environment-Kubus

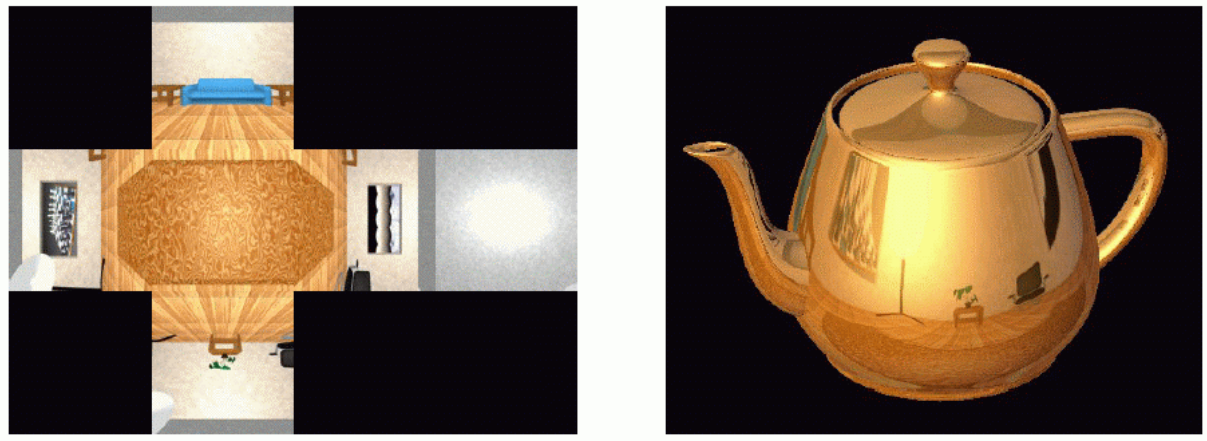


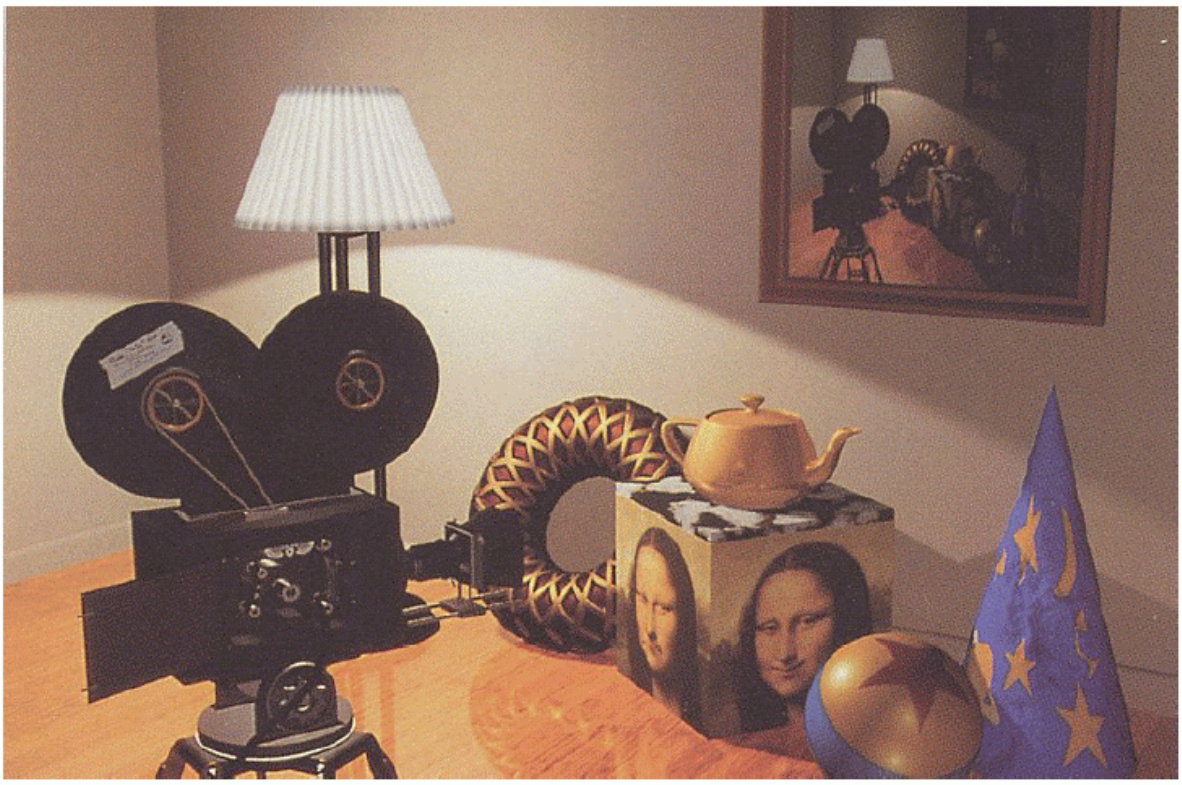
Verschiedene Varianten:

- Kugel (schon bei Blinn & Newell 1976)
- Würfel (Greene 1986)
- 2 Paraboloide ("dual-parabolisches Mapping", Heidrich & Seidel 1999) – gleichmäßigere Abtastung als beim Sphere Mapping, weniger Aliasing, einfache Matrixoperationen für die Berechnung

Beispiele:



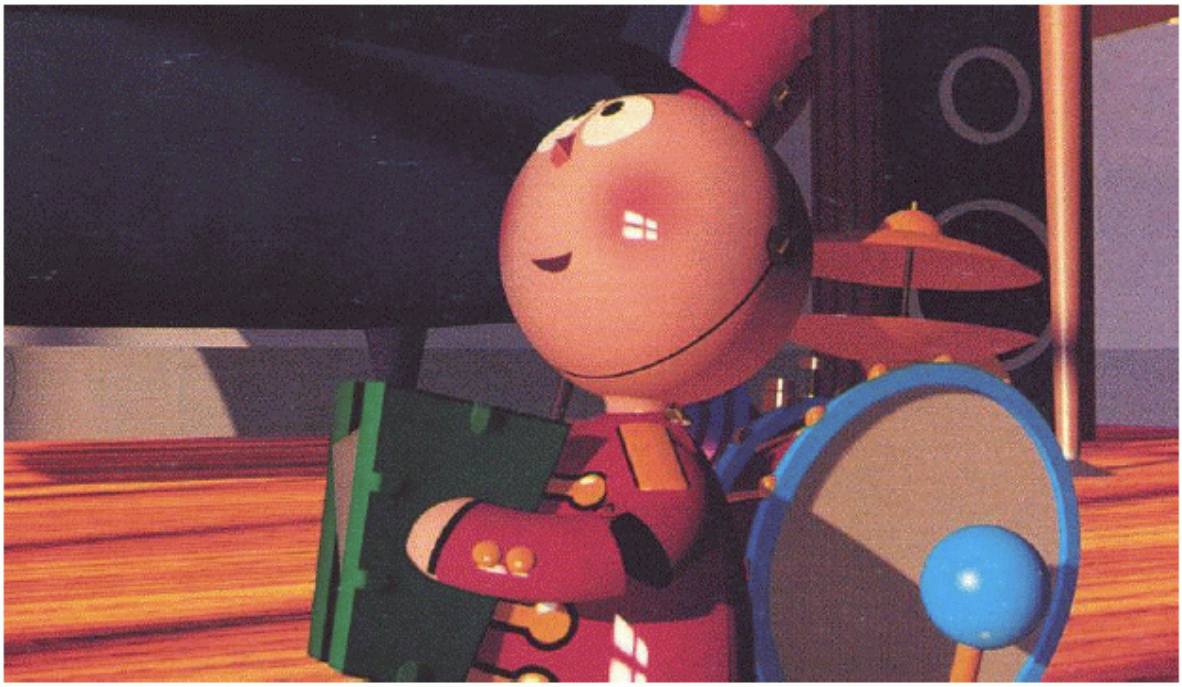




Reflection Mapping

Reflection-Mapping für Fenster-Rahmen

– Kein Ray-Tracing !!



Vorteile des Environment Mapping

- schnell und einfach zu berechnen
- liefert gute Ergebnisse, wenn die Textur z.B. den Himmel oder einen weit entfernten Horizont repräsentiert
- kann verwendet werden, um Reflexe von ausgedehnten Lichtquellen darzustellen

Probleme des Environment Mapping

- Die Reflektionsberechnung ist nur dann korrekt, wenn sich der Objektpunkt *P* im "Weltmittelpunkt" (Zwischenflächen-Mittelpunkt) *W* befindet. Je größer der Abstand zwischen *P* und *W*, desto stärker die Verzerrungen
- Ist die Environment Map schlecht parametrisiert, können erhebliche Aliasing-Probleme auftreten
- Es wird keine Verdeckungsrechnung durchgeführt
- Szenenobjekte können sich nicht gegenseitig widerspiegeln
- Vorsicht vor Artefakten an den Kanten und "Nahtstellen" des Projektionskörpers und durch Interpolation