11. 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
$\Rightarrow$ nicht akzeptabel in Bezug auf Rendering-Zeit und Speicherplatz
$\Rightarrow$ 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*

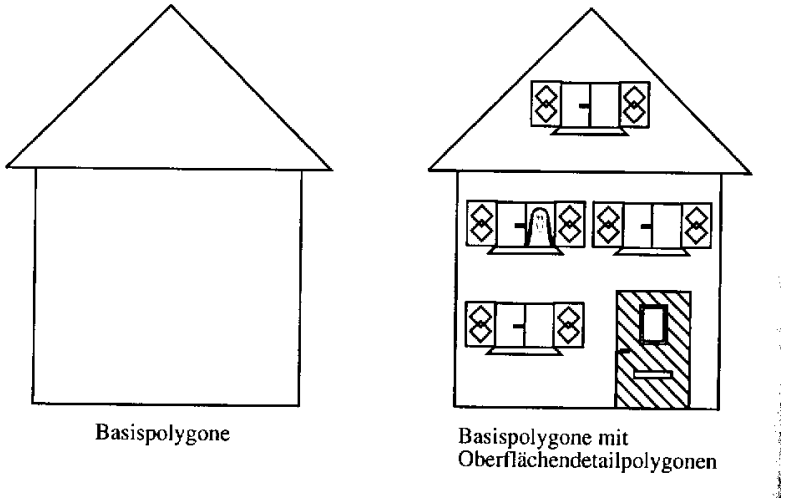
Geschichte von Texturen

- **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

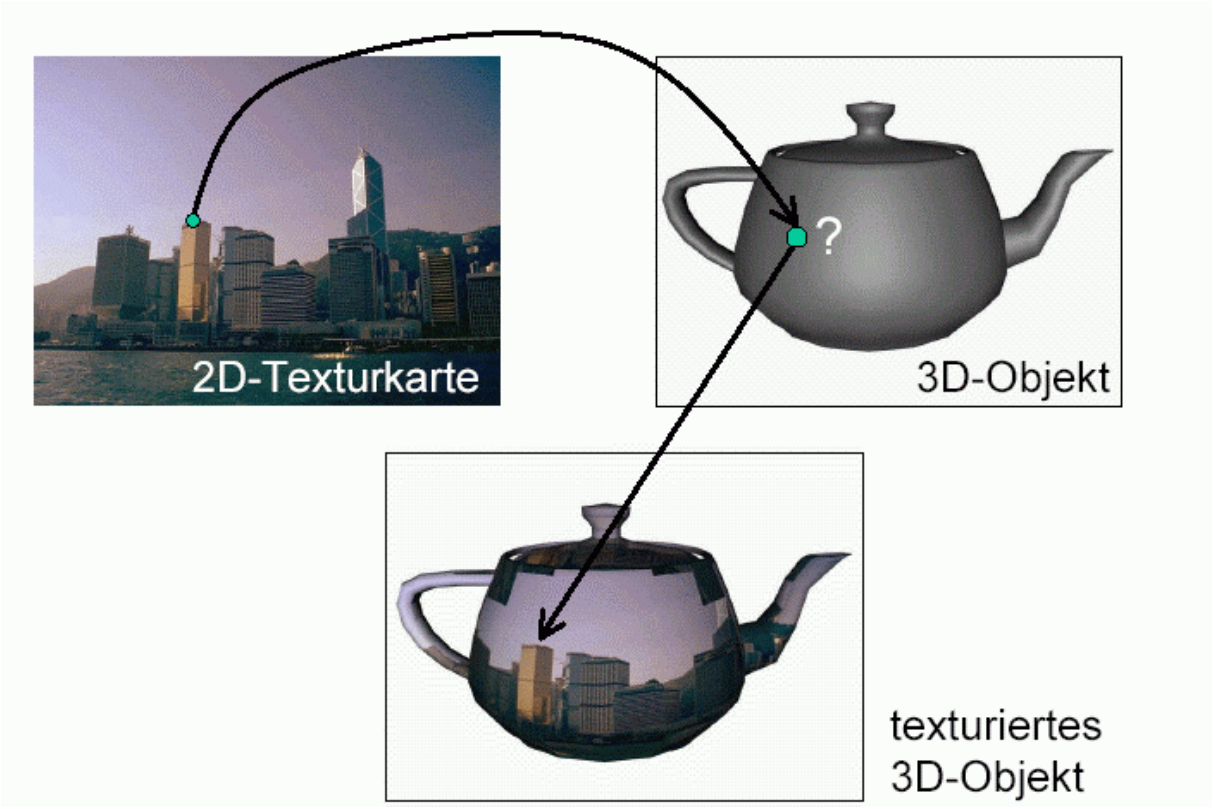
Basispolygone mit Oberflächendetailpolygonen

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

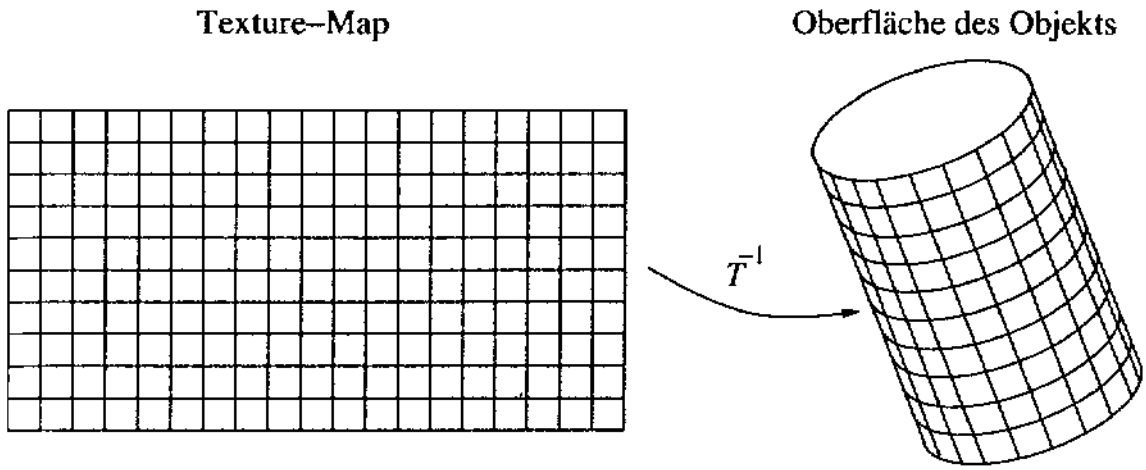
Alternative:

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?



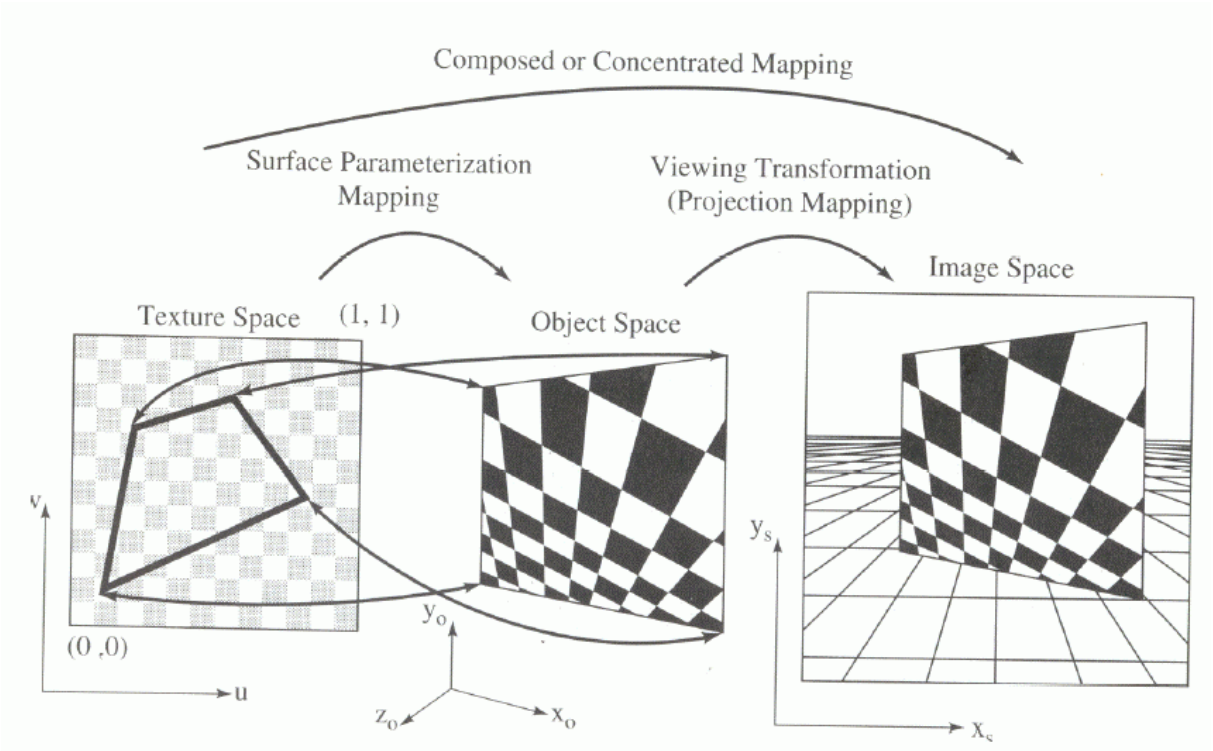
Das *Mapping* beschreibt, wie eine 2D-Textur bzw. ein Ausschnitt davon auf eine Fläche aufgebracht wird.
Beim Rendering muss jedoch das *inverse Mapping-Problem* gelöst werden, d.h. den bekannten (*x*, *y*, *z*)-Koordinaten eines Punktes der Oberfläche im Objektraum müssen Texel-Koordinaten (*u*, *v*)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

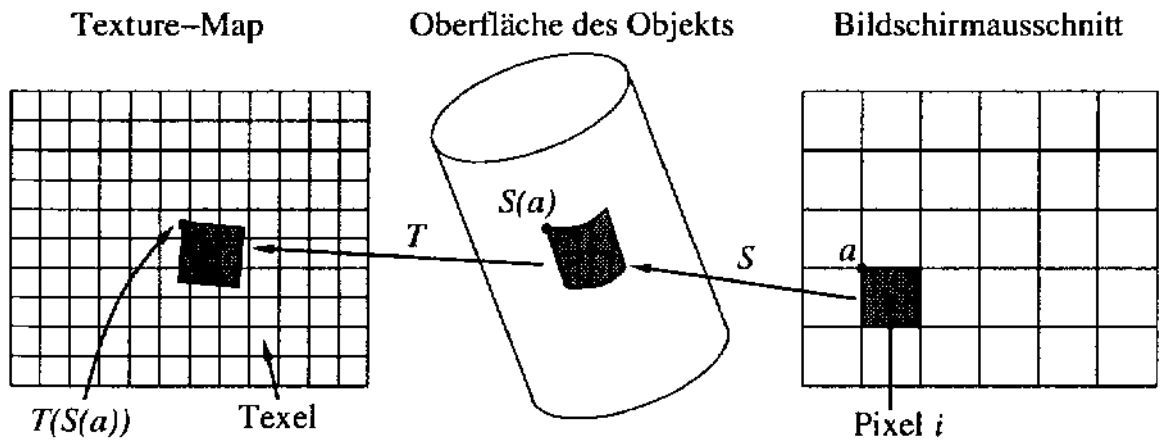


Insgesamt werden 2 Transformationen (Mappings) benötigt:

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

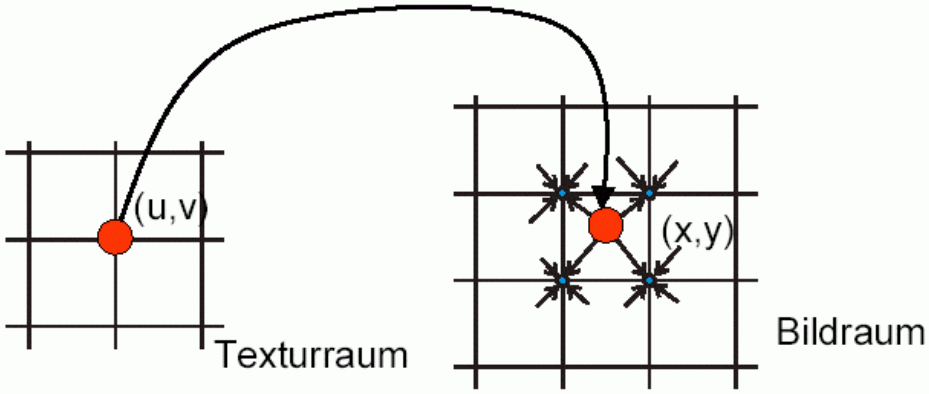
häufig werden beide Transformationen zu einer zusammengefasst (*composed mapping*, Texturabbildung vom Texturraum in den Bildraum).





Zusammengesetzte, inverse Abbildung:

Texturabbildung vom Texturraum in den Bildraum:



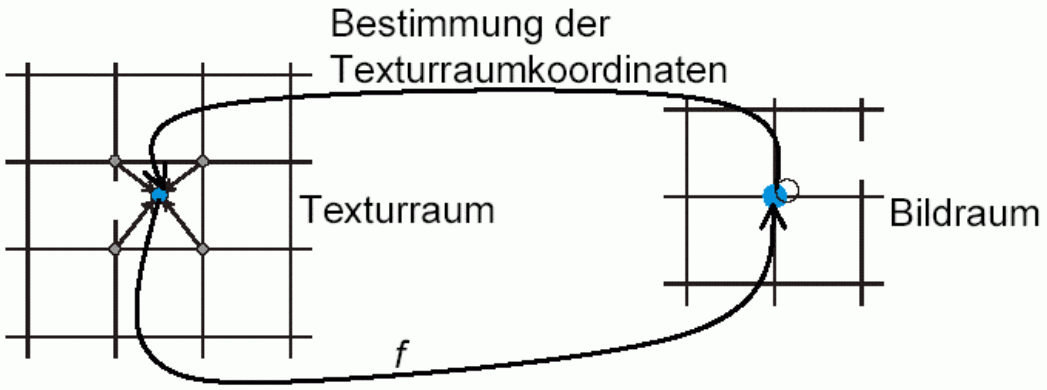
Der Farbwert des transformierten Pixels wird auf die benachbarten, ganzzahligen Bildraumkoordinaten aufgeteilt.

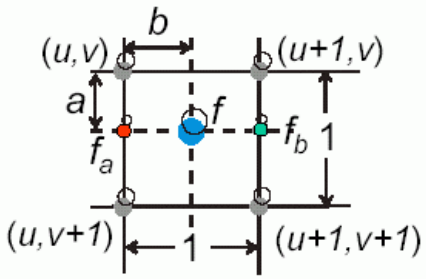
Einfacher Ansatz:
den Objektraum (und dortige, kompliziertere Strukturen) "umgehen" oder vereinfachen und die Texturabbildung (bzw. ihre Inverse) direkt mit einfachen Funktionen beschreiben

Varianten:

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

Inverse Texturabbildung :





Bilineare Interpolation:

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

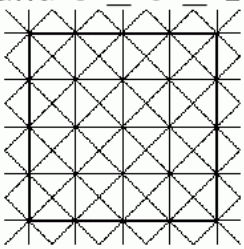
$$f_b = a\,f(u+1,v) + (1-a)\,f(u+1,v+1)$$

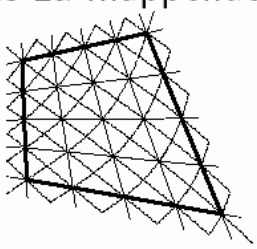
$$f = b\,f_a + (1-b)\,f_b$$

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

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

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

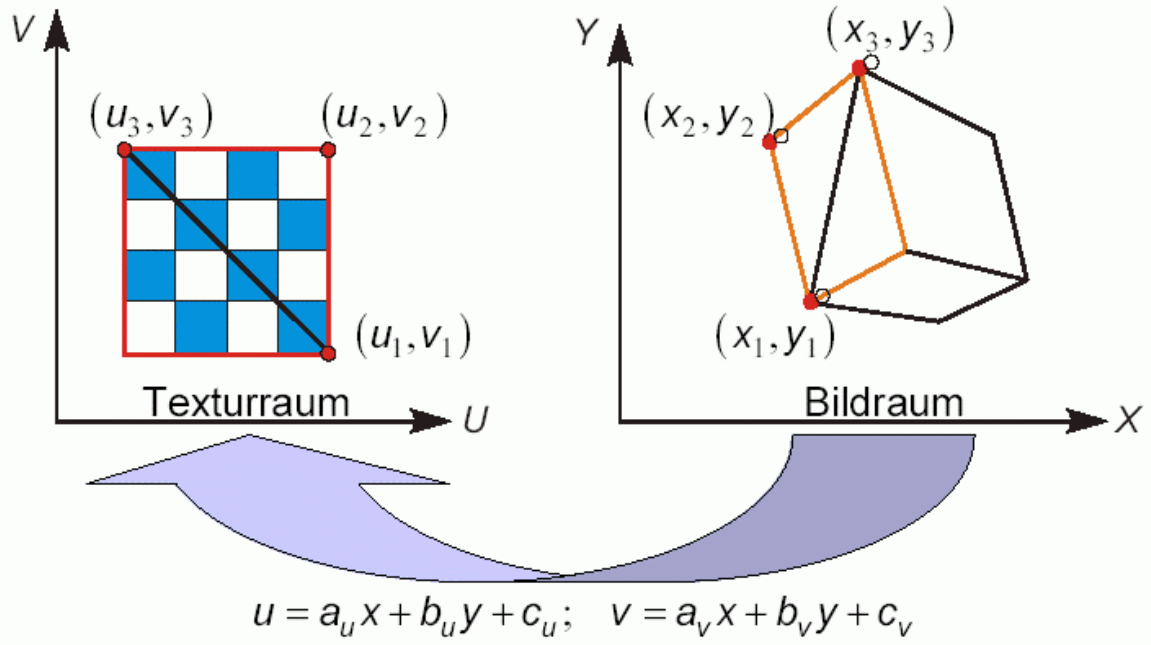




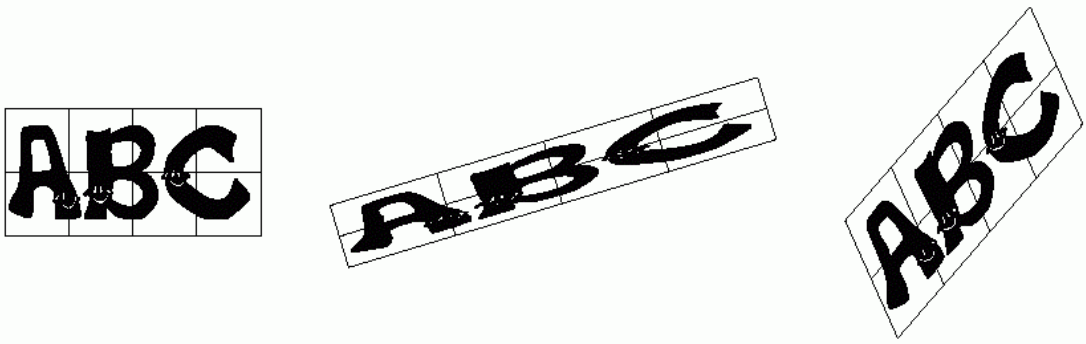
- vertikale und horizontale Geraden bleiben Geraden
- diagonale Geraden werden zu Kurven
- Kompositionen bilinearer Transformationen sind biquadratisch, schwer handhabbar

Affine Texturabbildung:

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

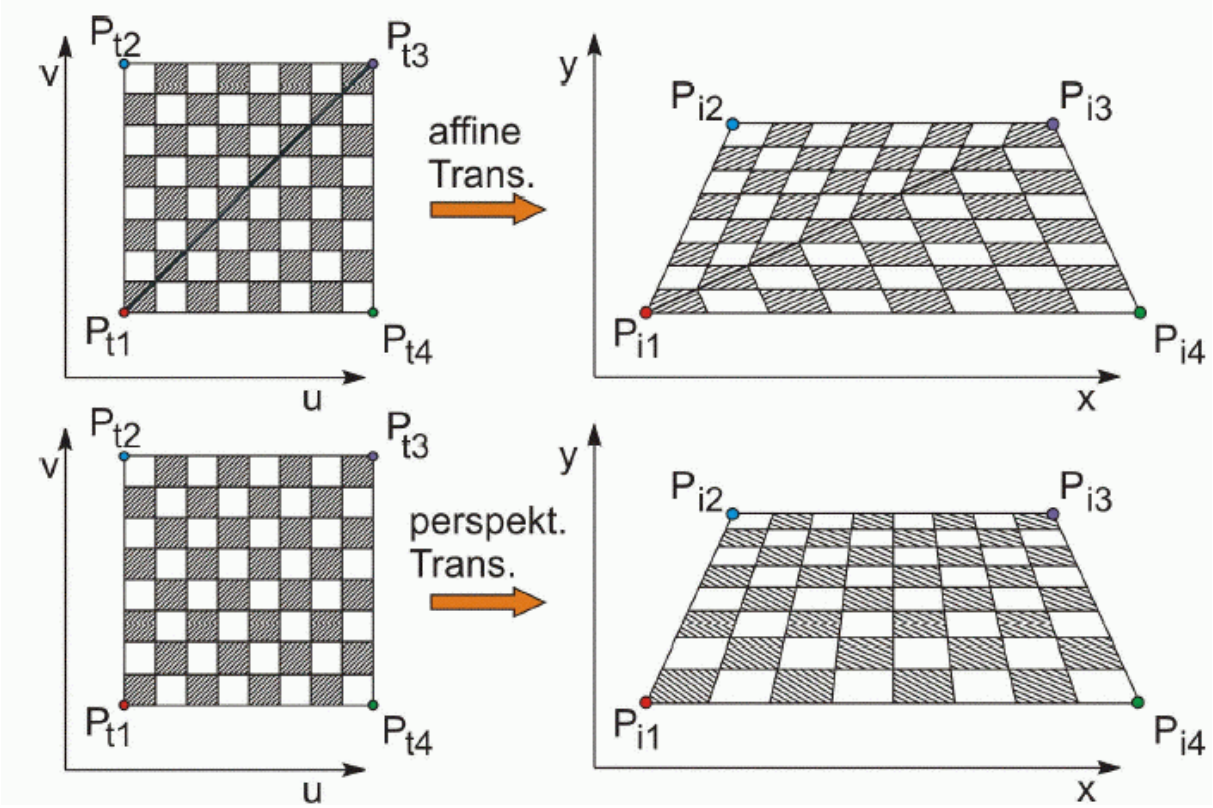


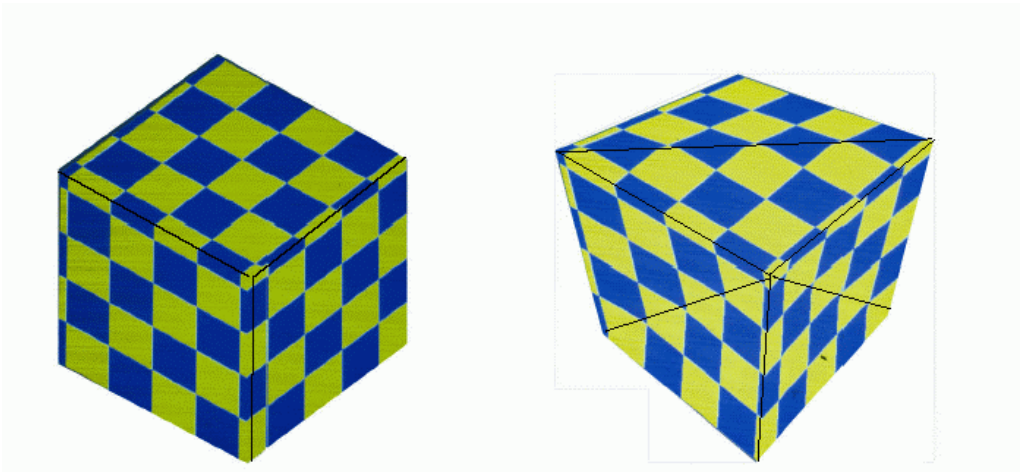
- Affine Mapping-Transformation
  - Skalierungen, Rotationen, Translationen, Scherungen und Kombinationen davon
  - erhalten parallele Linien
  - Dreiecke → Dreiecke, Rechtecke → Parallelogramme



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:

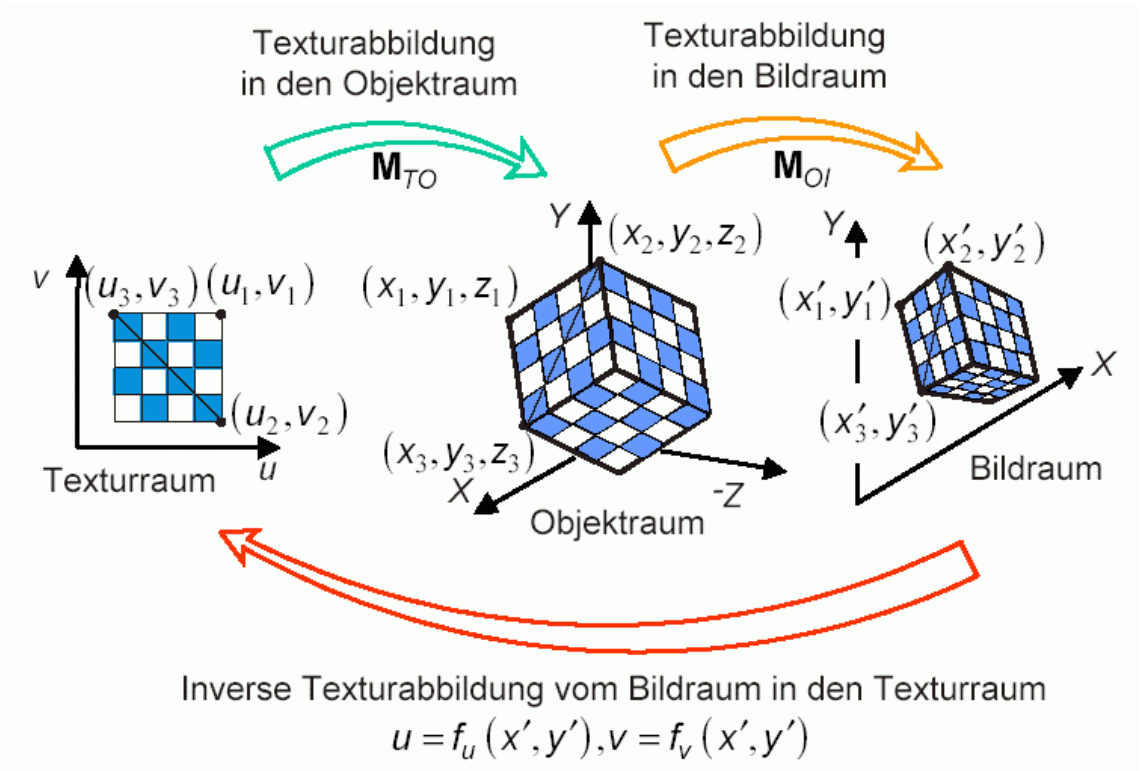




Affine Texturabbildung auf parallel projiziertes Objekt

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:



- 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.

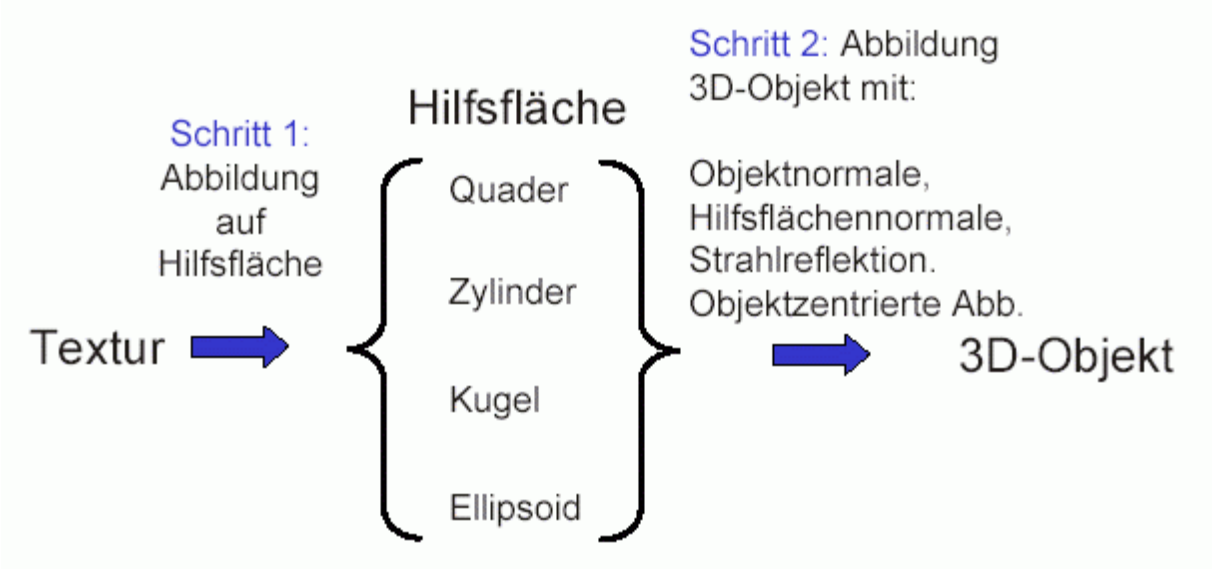
- Besteht ein Objekt aus sehr wenigen Polygonen, so kann ein Anwender die Zuordnung der Polygone zur Textur-Referenzfläche u.U. noch interaktiv vornehmen
- bei komplexeren Objekten ist dies unmöglich

$\Rightarrow$ *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.

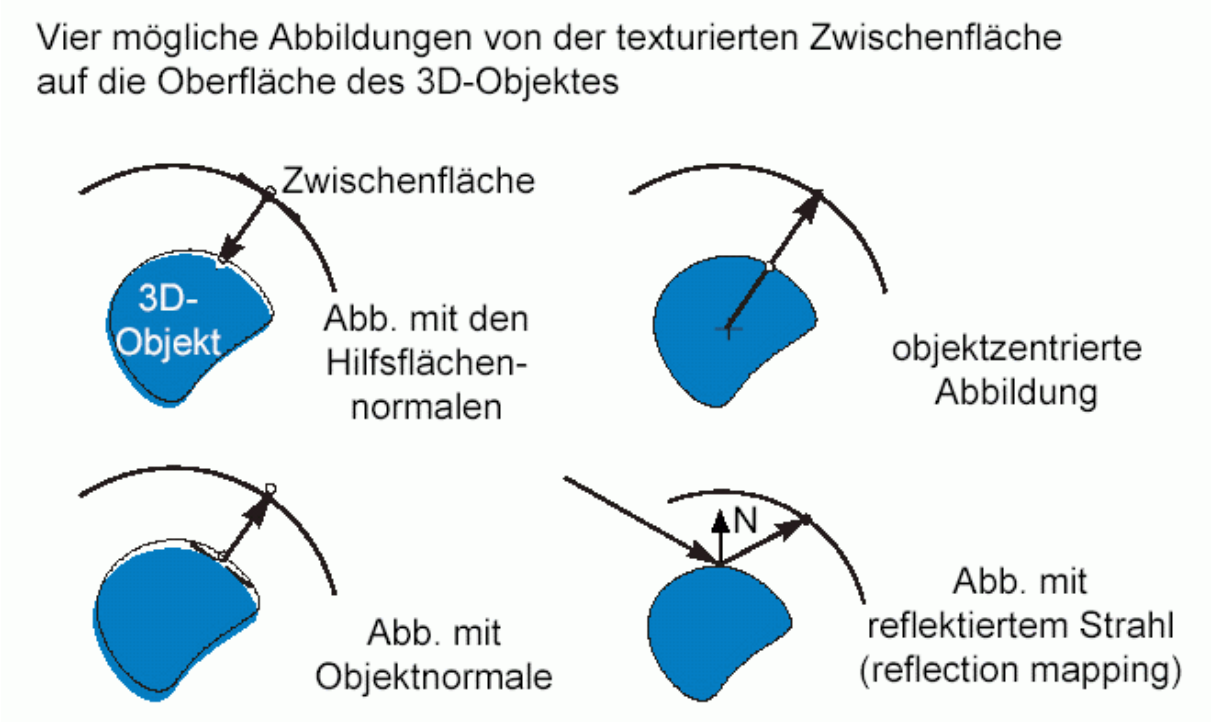
$$\begin{array}{ccccc} & & S & & O \\ (u,v) & \longrightarrow & (x_i, y_i) & \longrightarrow & (x_o, y_o, z_o) \\ \text{texture space} & \longrightarrow & \text{intermediate space} & \longrightarrow & \text{object space} \end{array}$$



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:



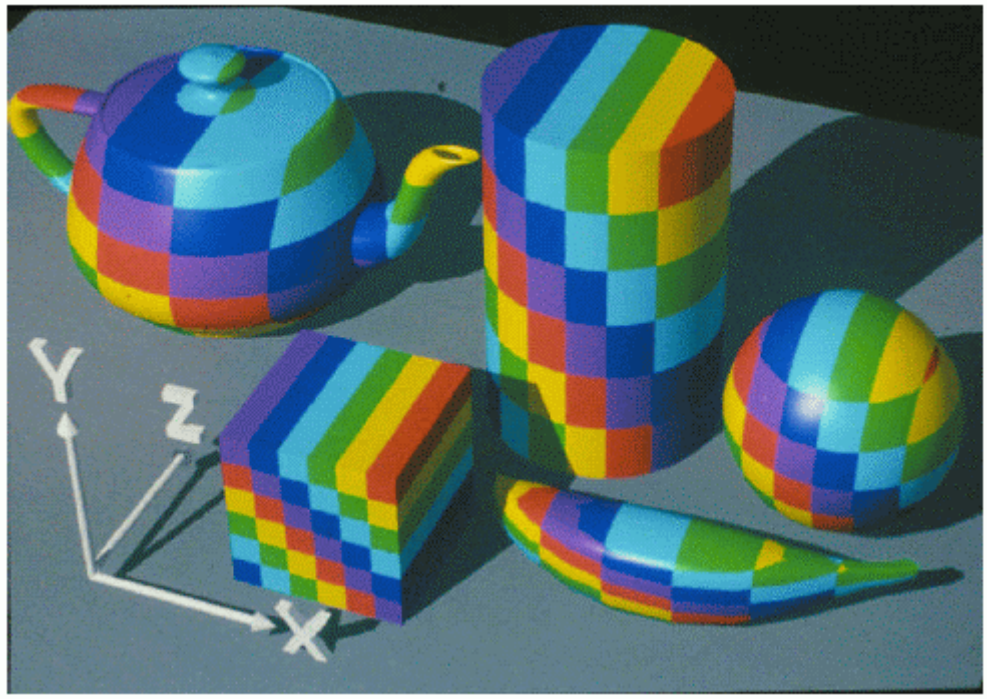
man unterscheidet nach Art der Zwischenfläche:

- 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:

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

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

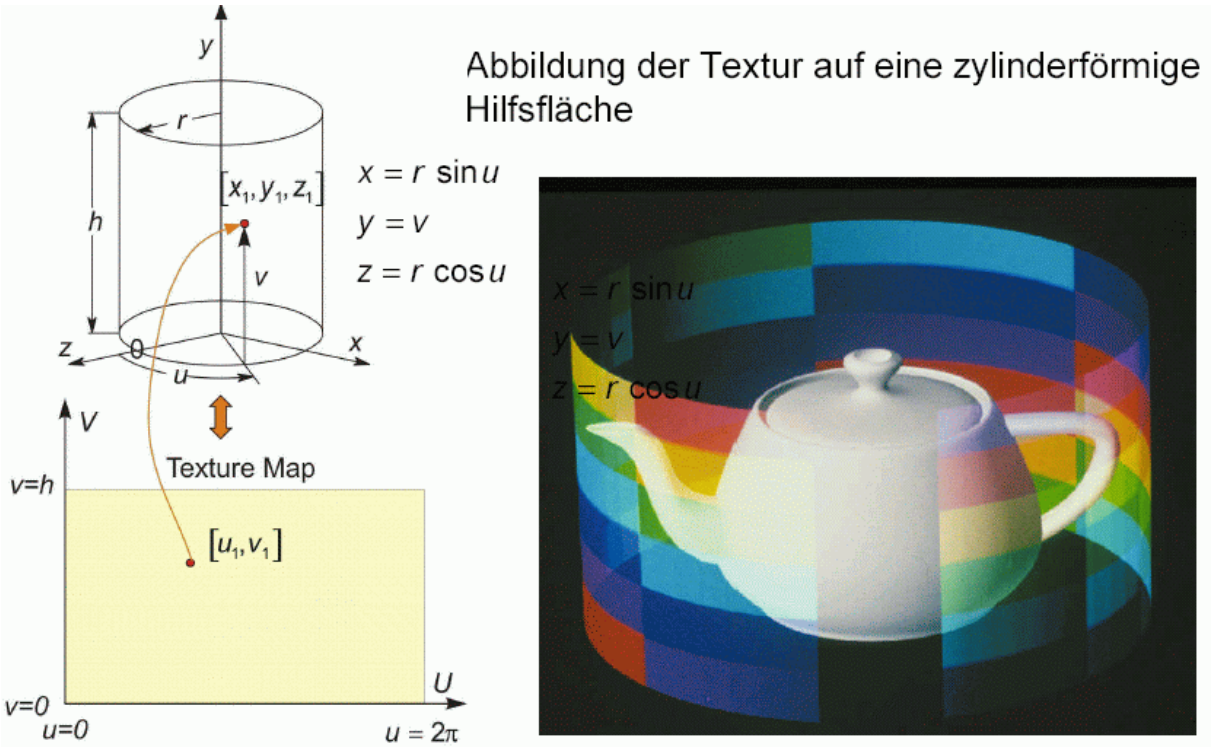
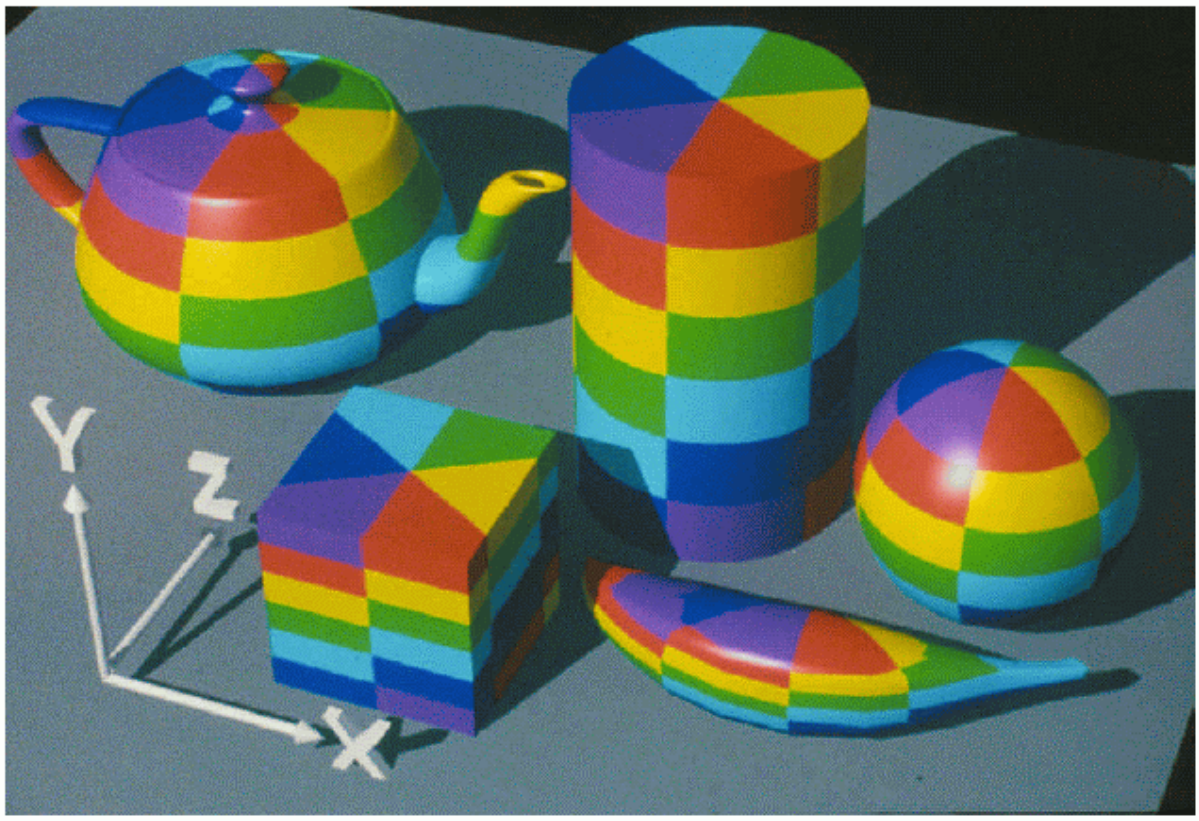


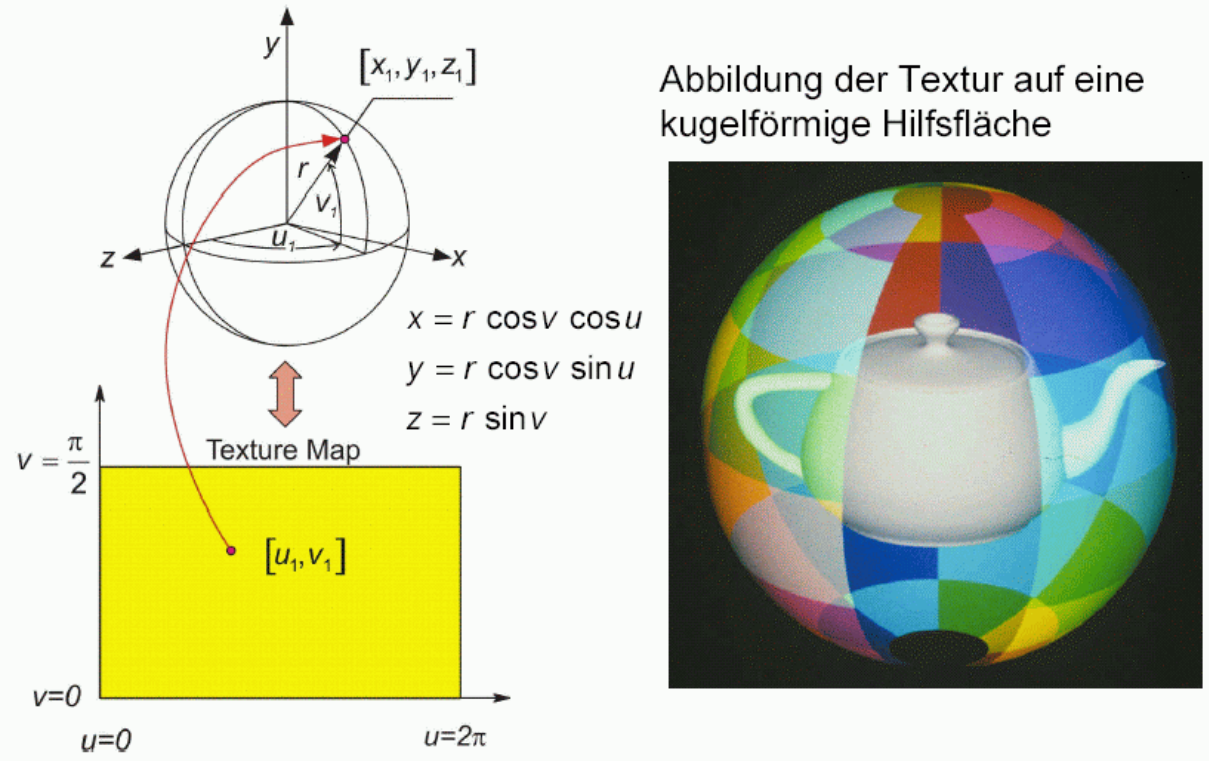
Abbildung der Textur auf eine zylinderförmige Hilfsfläche
y
r
h
v
z
0
x
u
x_1, y_1, z_1
$x = r \sin u$
$y = v$
$z = r \cos u$
V
Texture Map
v=h
u_1, v_1
v=0
u=0
U
u = 2π
$x = r \sin u$
$y = v$
$z = r \cos u$

Anwendung:

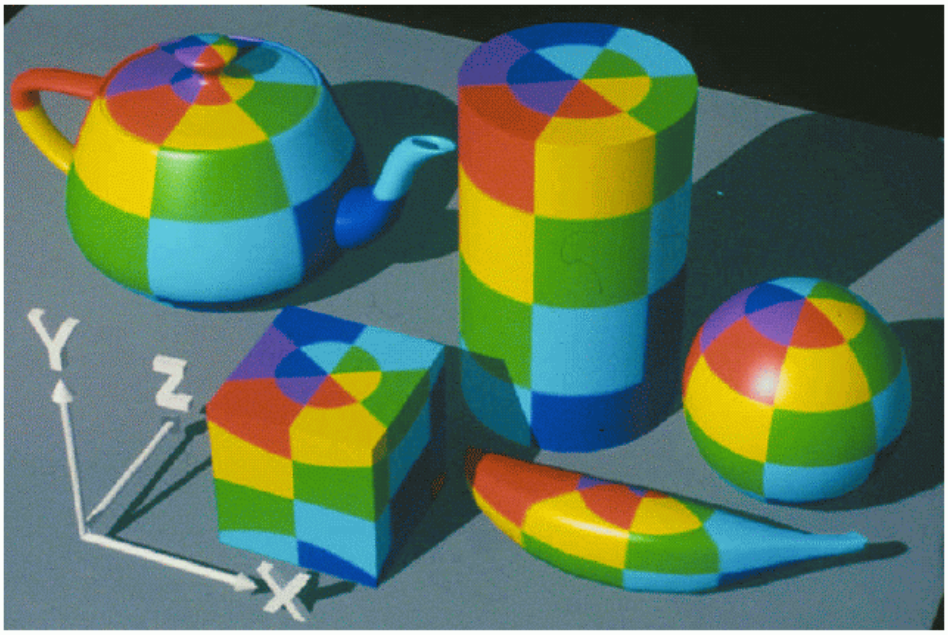

Y
Z
X

Kugel-Mapping (*sphärisches Mapping*):

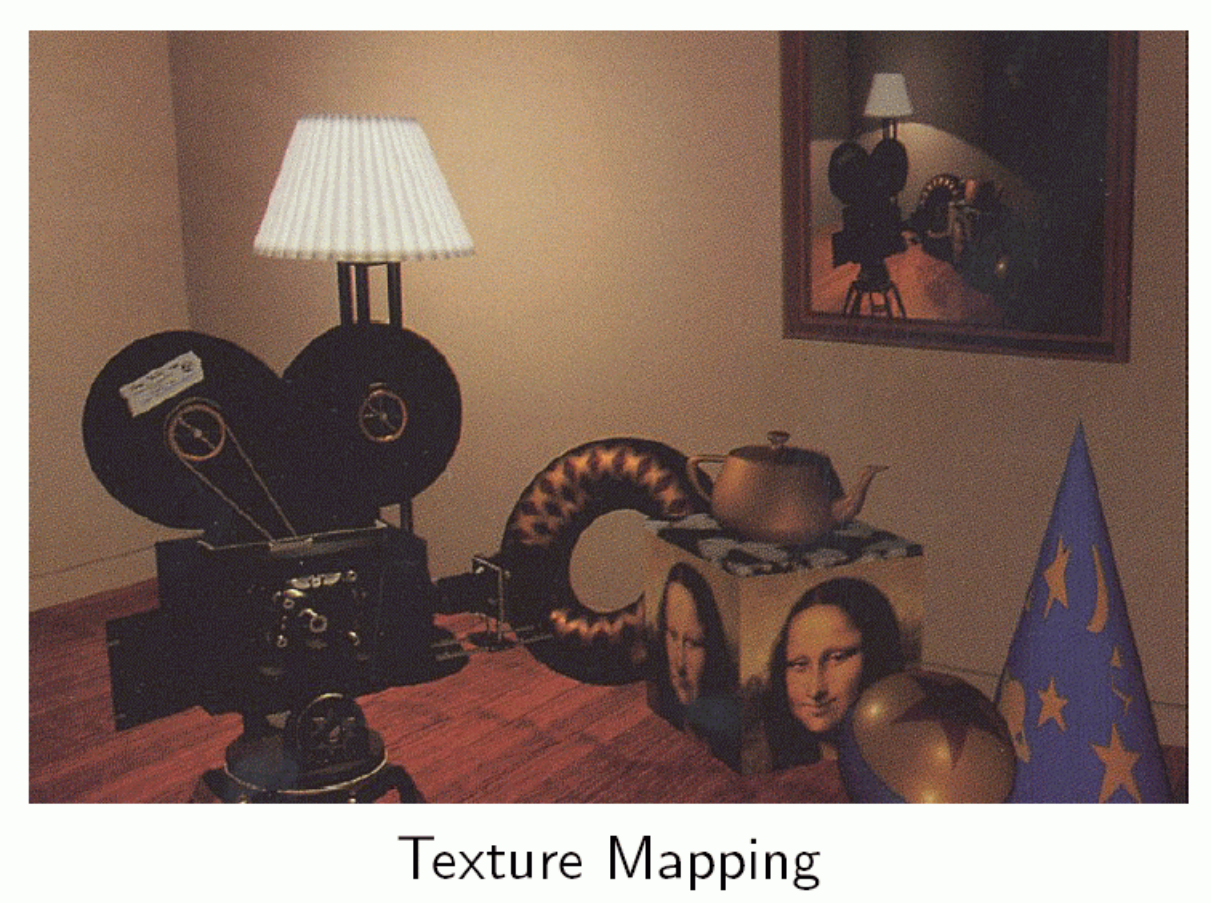
- Abbildung auf eine Kugel (bzw. 2 Halbkugeln) und dann auf das Objekt
- stereografische Projektion für die Abb. auf die Halbkugeln
- Diskontinuität entlang des Äquators
- generell keine verzerrungsfreie Abb. einer ebenen Fläche auf eine Kugeloberfläche und umgekehrt möglich



Anwendungsbeispiel:



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



Texture Mapping

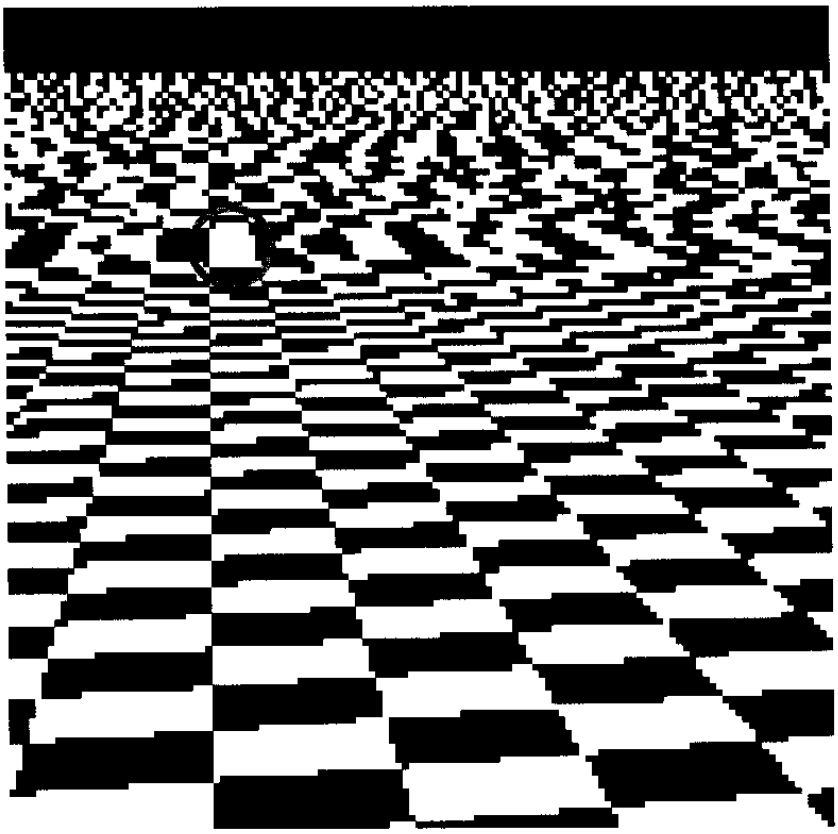
Probleme bei Verwendung von Rasterbildern als Texturen im Texture Mapping:

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

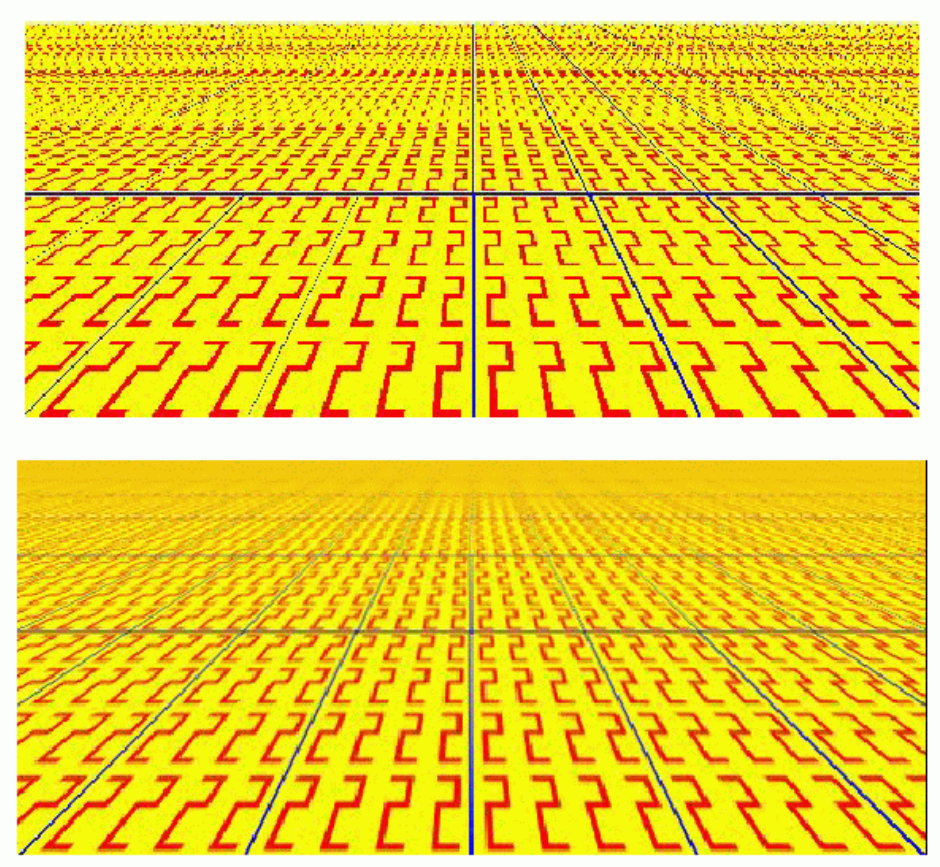
Abhilfe:

- 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:



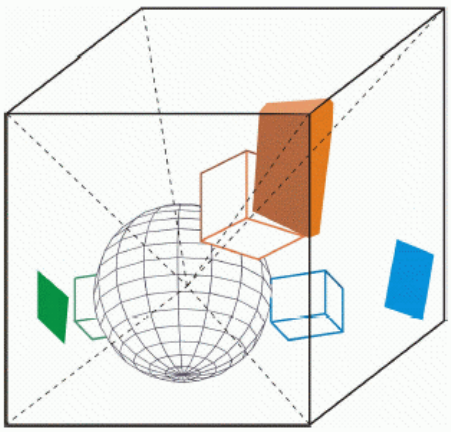
Environment Mapping / Reflection Mapping

= 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 $\Rightarrow$ die einfallende Beleuchtung kann für ein Objekt vorberechnet und in einer 2D-Textur, der *Environment Map*, gespeichert werden.

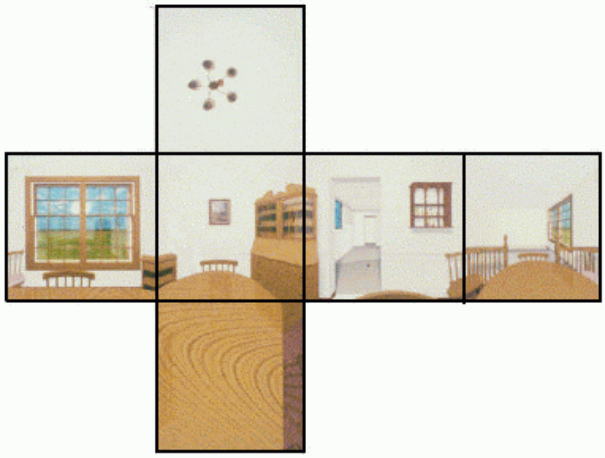
Realisierung:
Das reflektierende Objekt wird von einer virtuellen Zwischenfläche (Kugel, Box...) (wie beim two-pass texture mapping) umgeben, auf deren Innenseite die Szenenumgebung als 2D-Textur (Environment Map) aufgetragen ist.
Einem Punkt *P* auf der Objektoberfläche werden dann Texturkoordinaten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

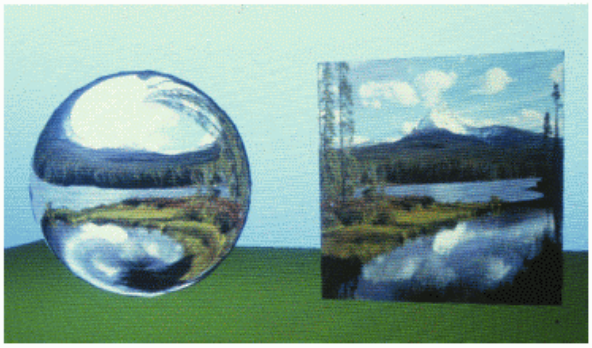
1. Schritt: Projektion der Szene vom Zentrum des zu texturiernden 3D-Objektes auf die sechs Seitenflächen des umgebenden Kubus



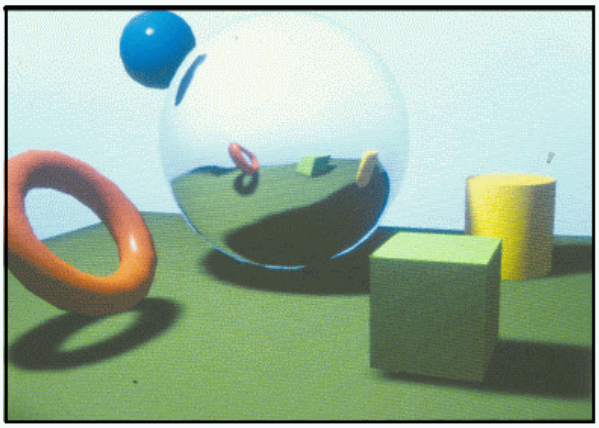
2. Schritt: Reflection Mapping

Beispiel:

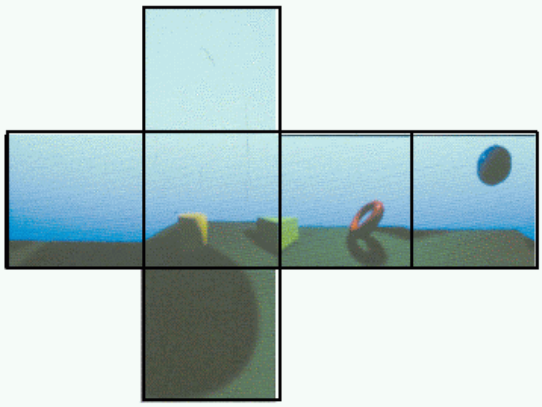




projizierte 3D-Szene



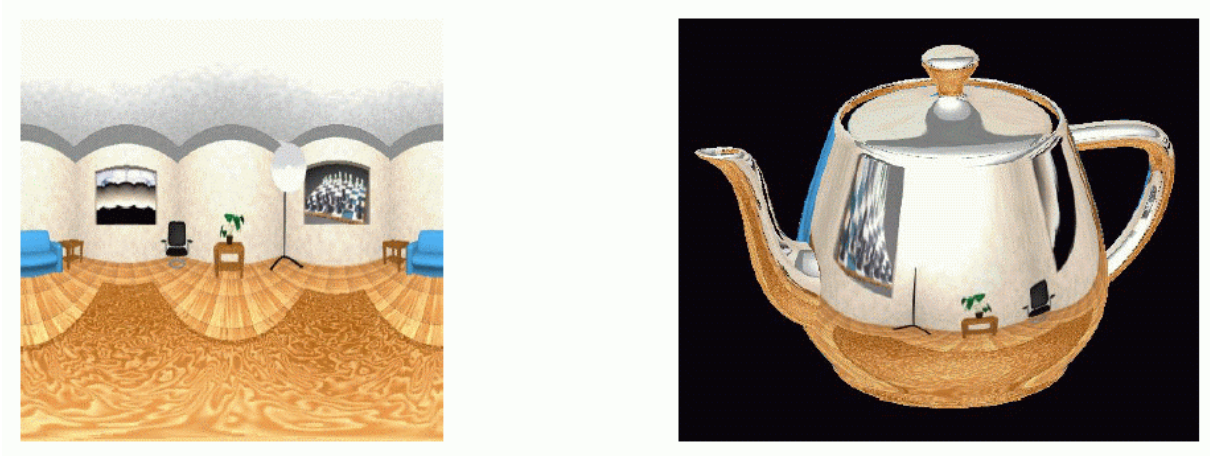
Abwicklung des Environment-Kubus

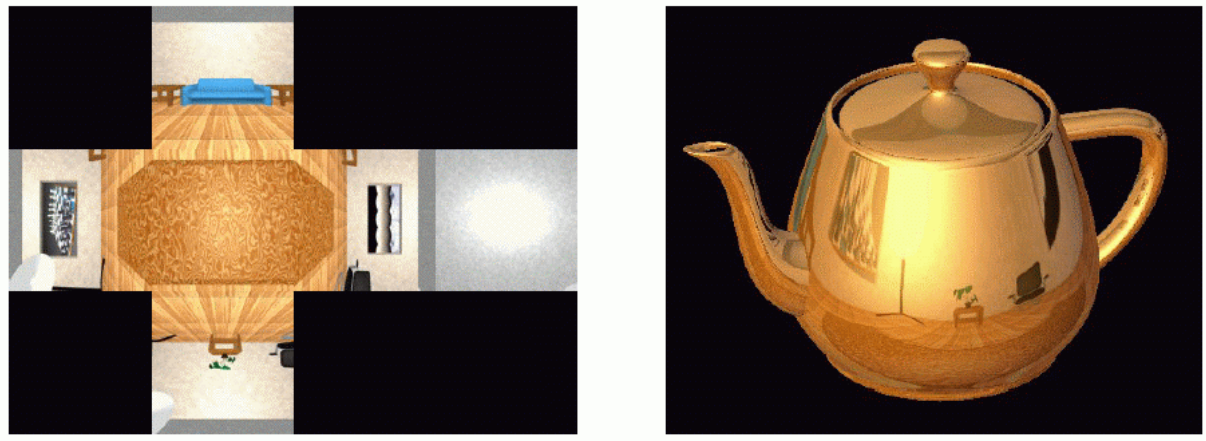


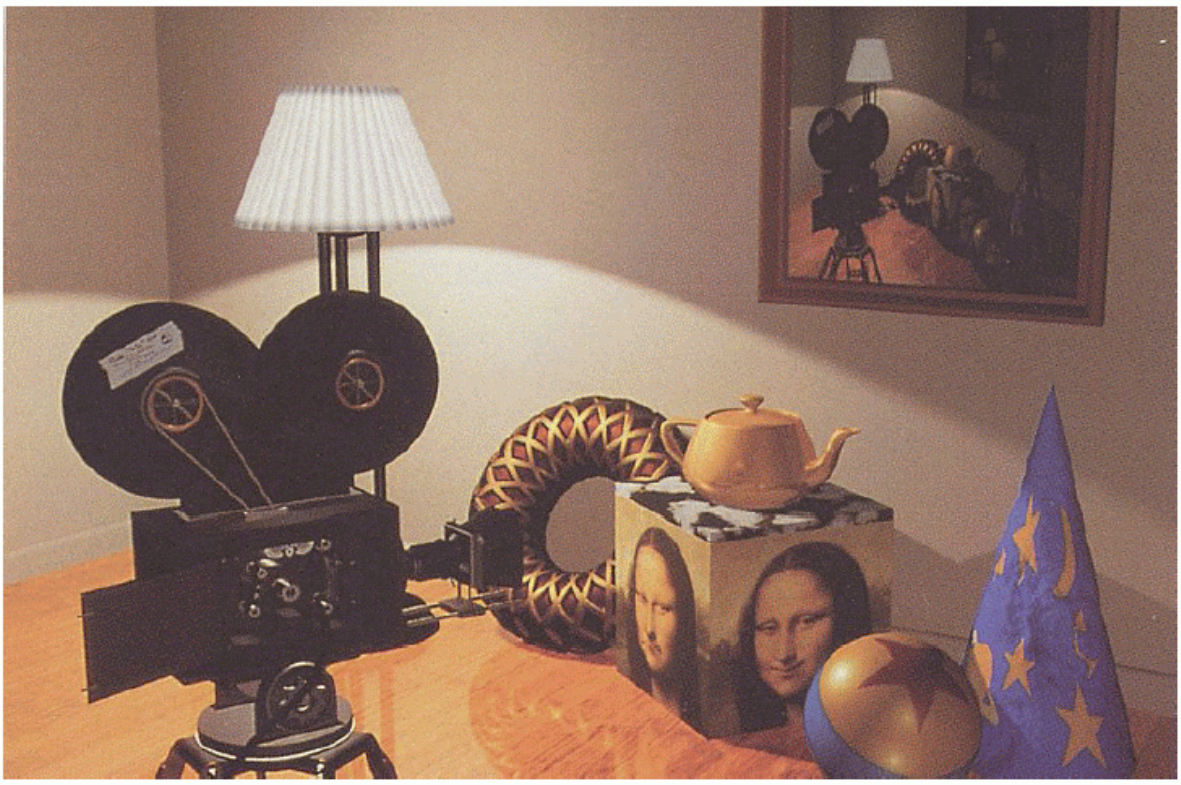
Verschiedene Varianten:

- 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:



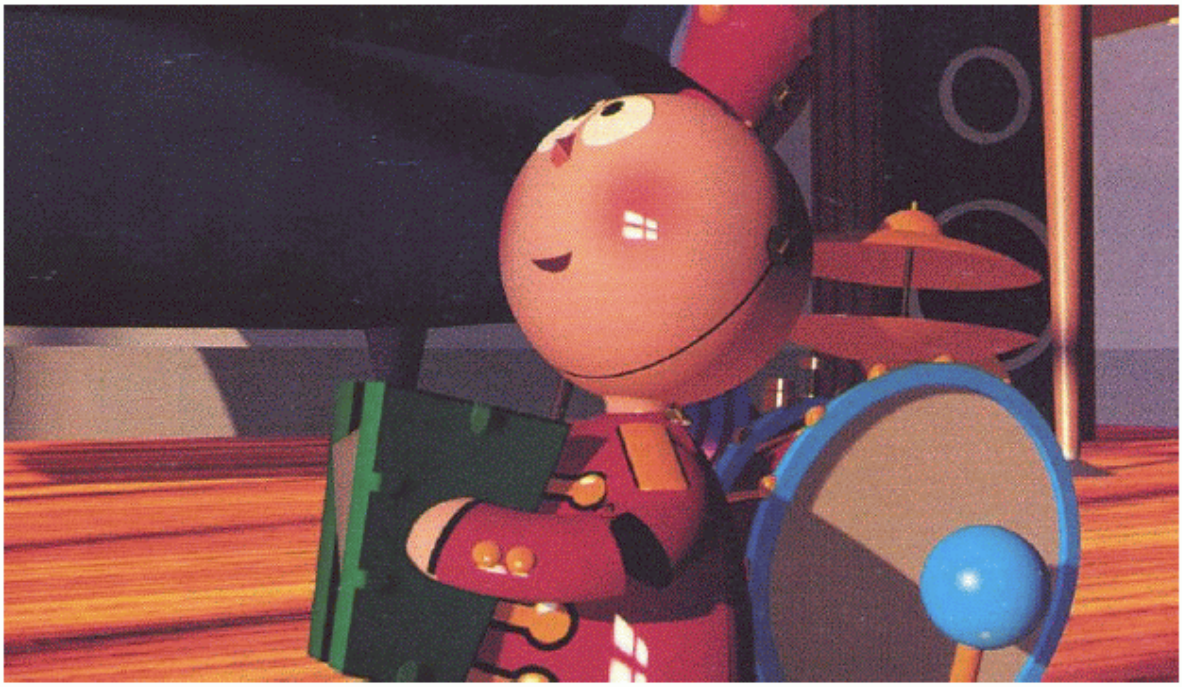




Reflection Mapping

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

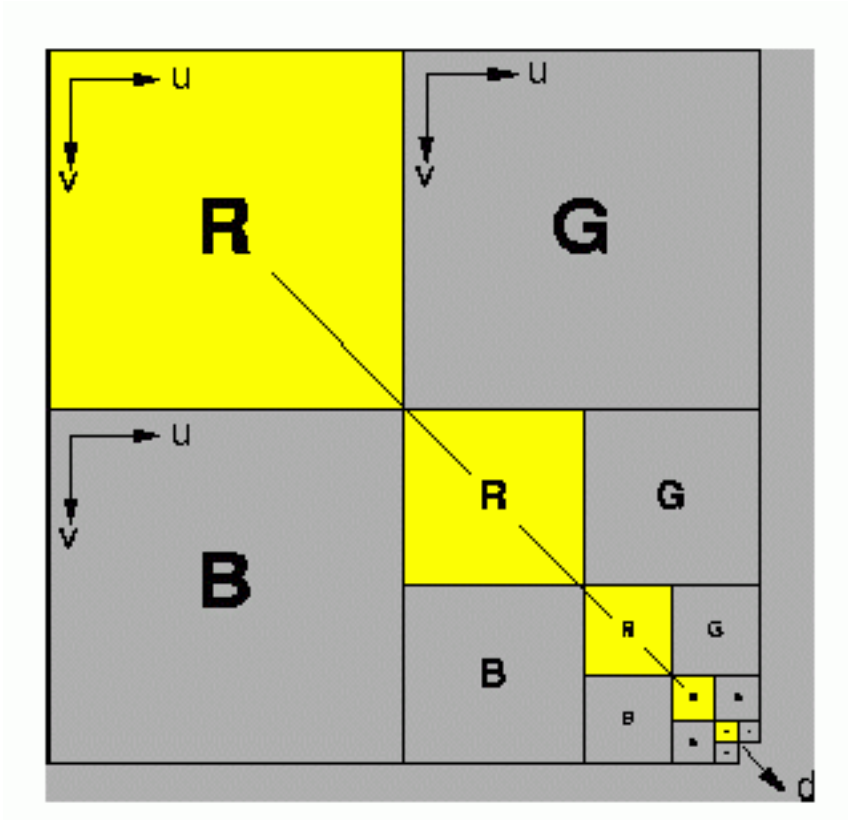
- Die Reflektionsberechnung ist nur dann korrekt, wenn sich der Objektpunkt *P* im "Weltmittelpunkt" (Zwischenflächen-Mittelpunkt) *W* befindet. Je größer der Abstand zwischen *P* und *W*, desto stärker die Verzerrungen
- Ist die Environment Map schlecht parametrisiert, können erhebliche Aliasing-Probleme 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

Techniken der Effizienzsteigerung bei 2D-Texturierung:

Mip-Mapping

MIP = "Multum in Parvo" = vieles auf kleinem Raum

- spezielle Texture-Mapping-Erweiterung, häufig bei Echtzeit-anwendungen, z.B. Spielen, verwendet
- LOD (level of detail) in Texturen
- Idee: berechne eine "Pyramide" gefilterter (vergröberter) Bilder im Voraus; Auswahl der Ebene, aus der ein Textur-pixel entnommen wird, abhängig von der Entfernung
- meist speichert eine Mip-Map C_{mip} eine quadratische Textur der Größe $n \times n$, wobei n eine Zweierpotenz ist, in fortwährend halbierten Auflösungsstufen



Auf der Stufe $d=0$ werden die Texturwerte direkt übernommen.

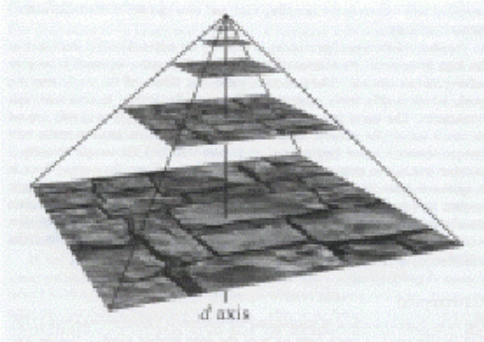
$$C^0_{mip}[i,j] = C[i,j],\ 0 \le i,j < 2^k.$$

Die übrigen Stufen d entstehen durch Filterung der jeweils vorhergehenden Stufe.

$$C^d_{mip}[i,j] = \frac{1}{4}\left(C^{d-1}_{mip}[2i,2j] + C^{d-1}_{mip}[2i+1,2j] + C^{d-1}_{mip}[2i,2j+1] + C^{d-1}_{mip}[2i+1,2j+1]\right)$$

$$1 \le d < k\text{-}1 \text{ und } 0 \le i,j < 2^{k-d}.$$

Auf der Stufe d der Texturhierarchie werden also 2^{2d} Texel der Originaltextur als ein einziges Texel dargestellt.



Diese Vorfilterung erfolgt nur einmal, unabhängig vom Betrachterstandpunkt.

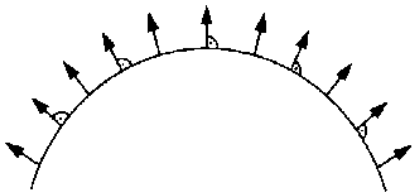
Bump Mapping

Mit den bisher beschriebenen Verfahren können nur Materialparameter variiert und Beleuchtungsergebnisse skaliert werden
⇒ beleuchtete Flächen sehen dadurch variantenreicher und realistischer aus, trotzdem wirken sie häufig noch zu glatt.

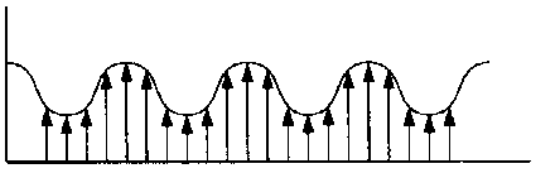
Gesucht: einfache Technik, mit der man Oberflächen rauh, runzlig, zerknittert, gefaltet, gekräuselt darstellen kann.

Exakte Modellierung von Unebenheiten: Aufaddieren eines 2D-Höhenfeldes zu einer ebenen Grundfläche (→ *Offsetfläche*)
⇒ oft zu rechenaufwändig
außerdem: in die Phong-Beleuchtungsgleichung gehen die exakten Positionen gar nicht ein; es kommt auf die *Normalenvektoren* an!

⇒ Idee des *Bump-Mapping* (Blinn 1978):
für kleine Unebenheiten reicht es aus, die Visualisierung mit der Originalgeometrie durchzuführen, bei der Beleuchtungsgleichung aber die Normalen der uneben gemachten Oberfläche zu verwenden.

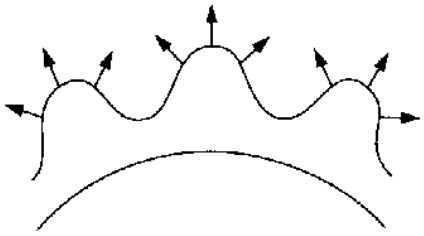


(a) Originalfläche **P**(u) mit Normalen **N**(u)



(b) Bump Map h(u)

(c) Offsetfläche **P'**(u)



(d) Perturbierte Normalen **N'**(u)

Resultat: scheinbare Oberflächenverformung im kleinen Maßstab durch Verwendung der abgelenkten Normalen-vektoren

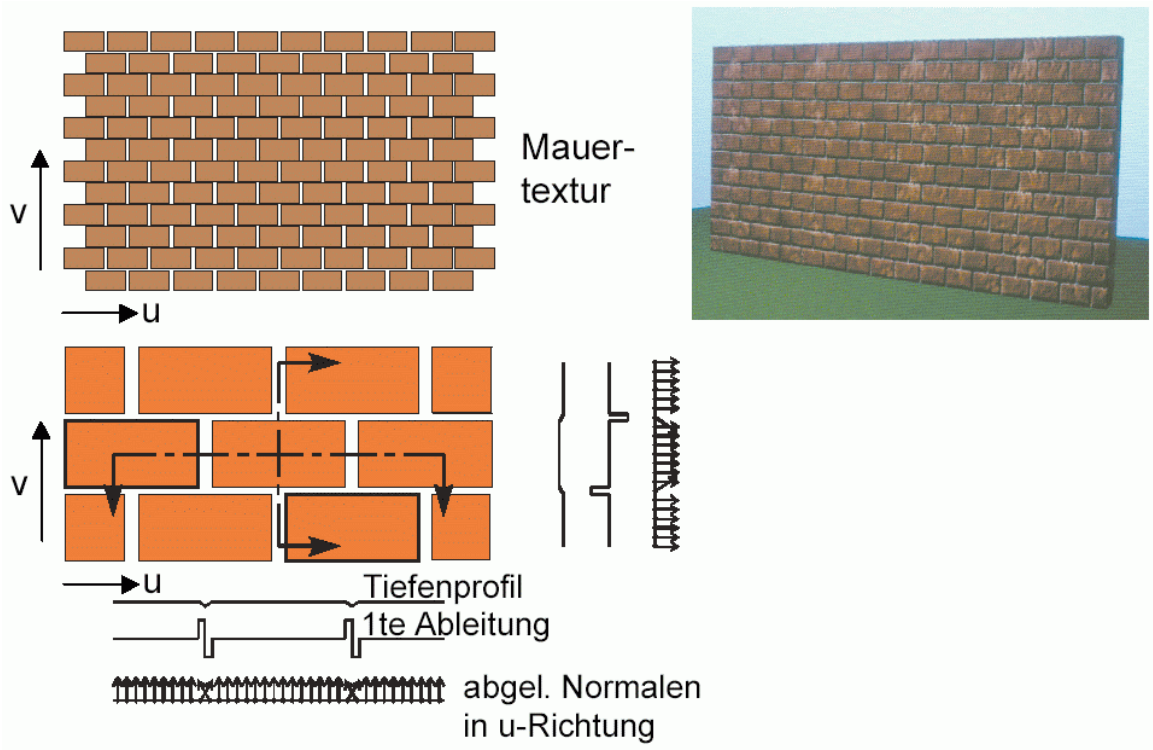
Bump Map = Feld mit Offset-Werten, die genutzt werden, um einen Punkt auf der Oberfläche nach oben oder unten zu versetzen (nur für die Bestimmung der Normalen)

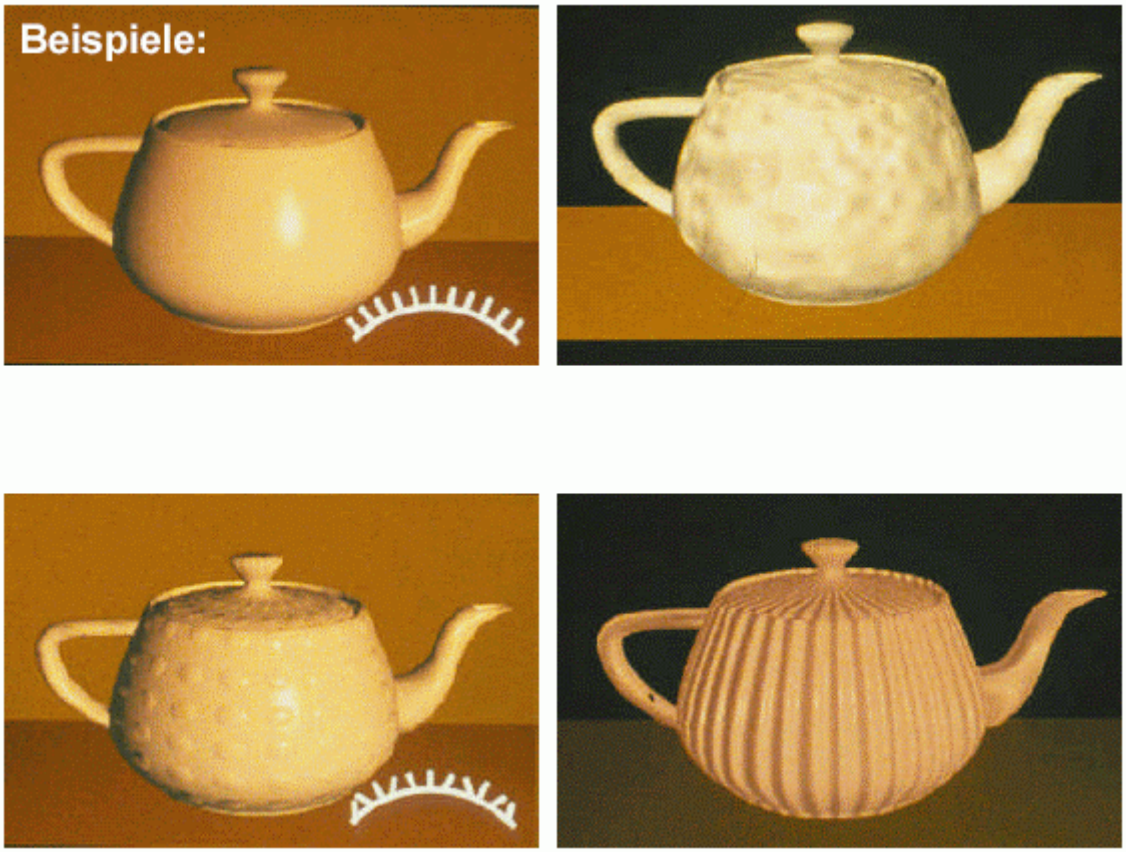
Den gesuchten Normalenvektor bestimmt man als Kreuz-produkt der Richtungsableitungen der Offsetfläche in einem gegebenen Punkt
(Formeln z.B. bei Krömker 2001).

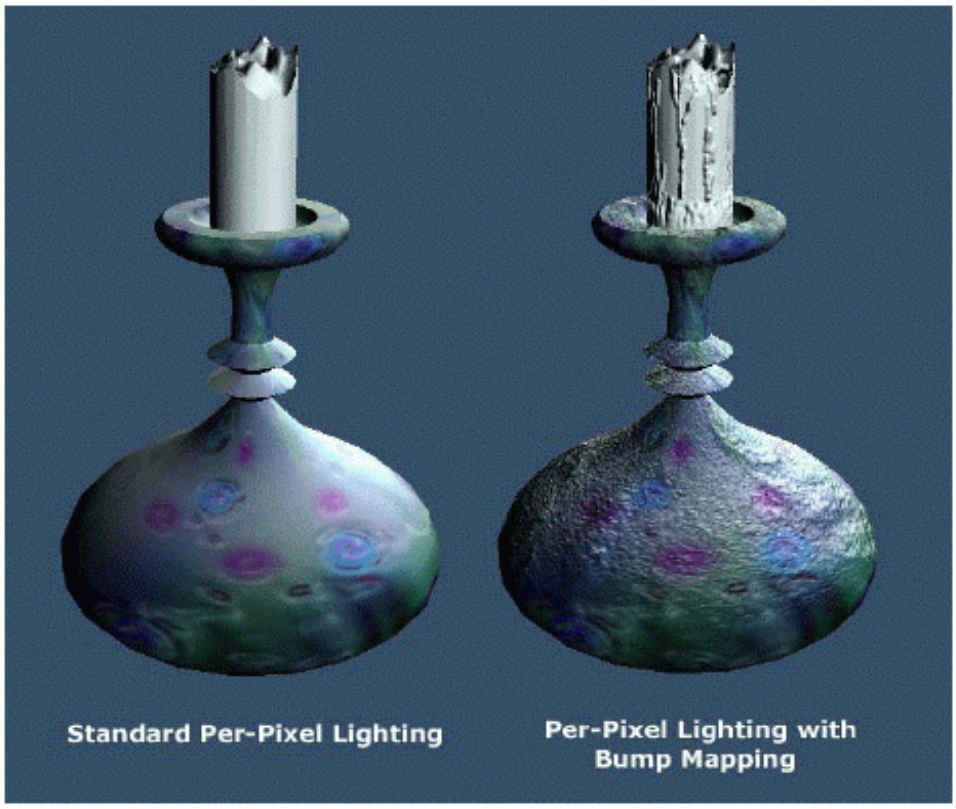
Beachte:

- Bump Mapping kann nur mit denjenigen Beleuchtungs-verfahren kombiniert werden, die eine explizite Beleuchtungsrechnung in jedem Flächenpunkt durchführen (Phong, Raytracing; **nicht** Gouraud).
- Seit 1997 gibt es Hardware-Implementationen des Bump Mapping.

Beispiele:







Bump Mapping

Bei größeren Unebenheiten: tatsächliche Berechnung der Offsetfläche und ihre Benutzung beim Rendering

"Displacement Mapping"

Überlagerung einer parametrischen Oberflächenfunktion

(x,y,z) = *f*(u,v) mit Hilfe eines Höhenfeldes (Displacementfunktion)

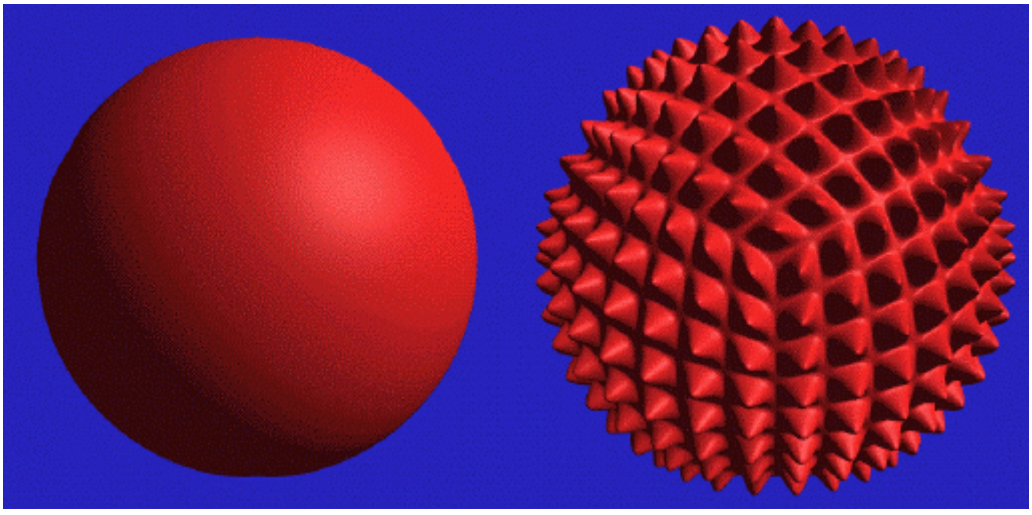
h = *g*(u,v).

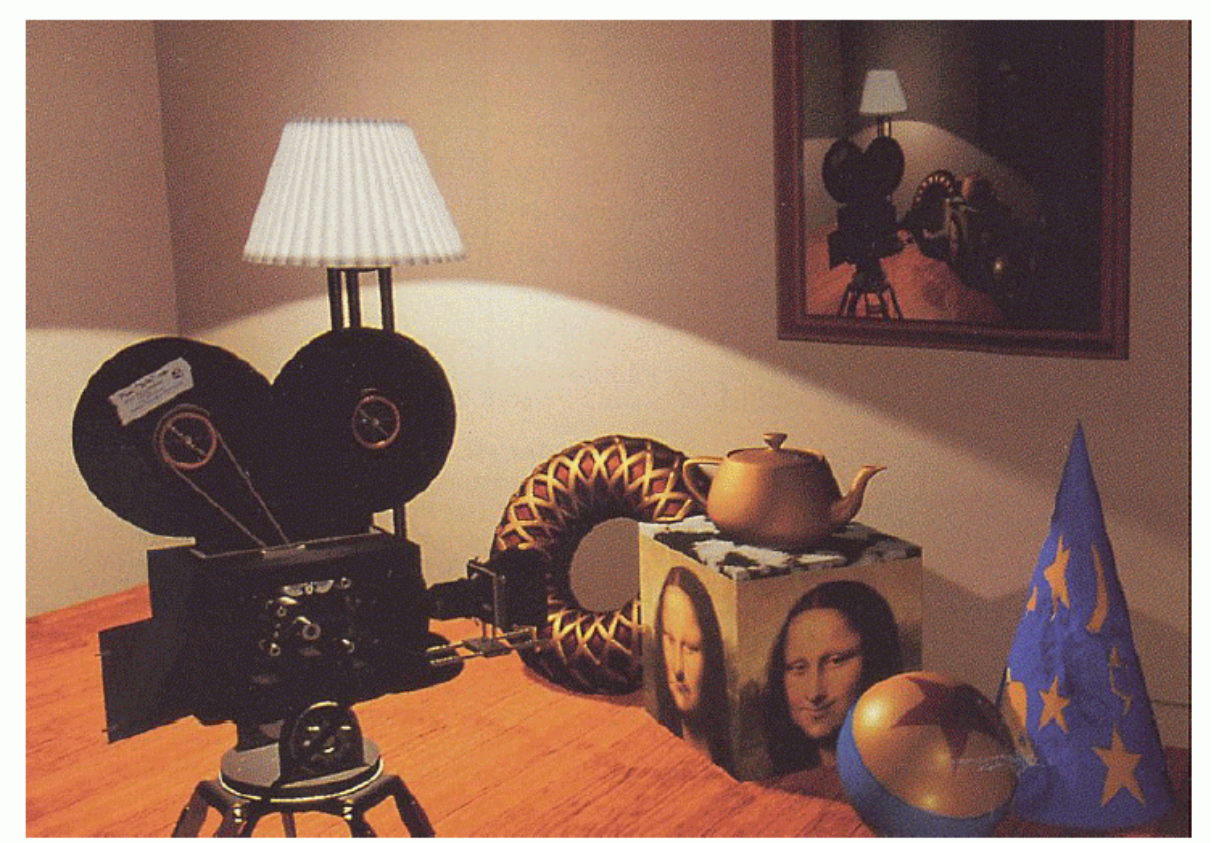
Wir erhalten:

$$f'(u,v) = f(u,v) + g(u,v) \cdot n(u,v)$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + h \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$$

Beispiel:





Displacement Mapping

alle bisher besprochenen Verfahren erzeugen Textur aus 2D-Zahlenfeldern: "diskrete Texturen"

Vorteile diskreter Texturen:

- einfache Erzeugung auch komplexer, realitätsnaher Muster (Foto, Zeichnung... dann einscannen)
- wenige, etablierte Verfahren; z.T. schon hardwaregestützt

Nachteile:

- Texturen mit hoher Auflösung haben hohen Speicherbedarf
- Beim Vergrößern von Bildern treten Artefakte auf
- Fortsetzung (Aneinandersetzen) von Texturen ist oft problematisch
- der der Textur zugrundegelegte Kontext (Sonnenstand, Schattenwurf...) stimmt häufig nicht mit der zu modellierenden Szene überein. Die Suche nach passenden Vorlagen kann aufwändig sein
- beim Mapping auf beliebige Flächen treten Verzerrungen und infolgedessen Abtast-Probleme (Aliasing) auf.

Alternative:

Prozedurale Texturen

(vgl. "prozedurale Modelle")

Bei jedem Aufruf eines Textur-Elements wird eine mathematische Formel bzw. ein Algorithmus ausgewertet.

Beispiel: Fractional Brownian Motion.

Vorteile:

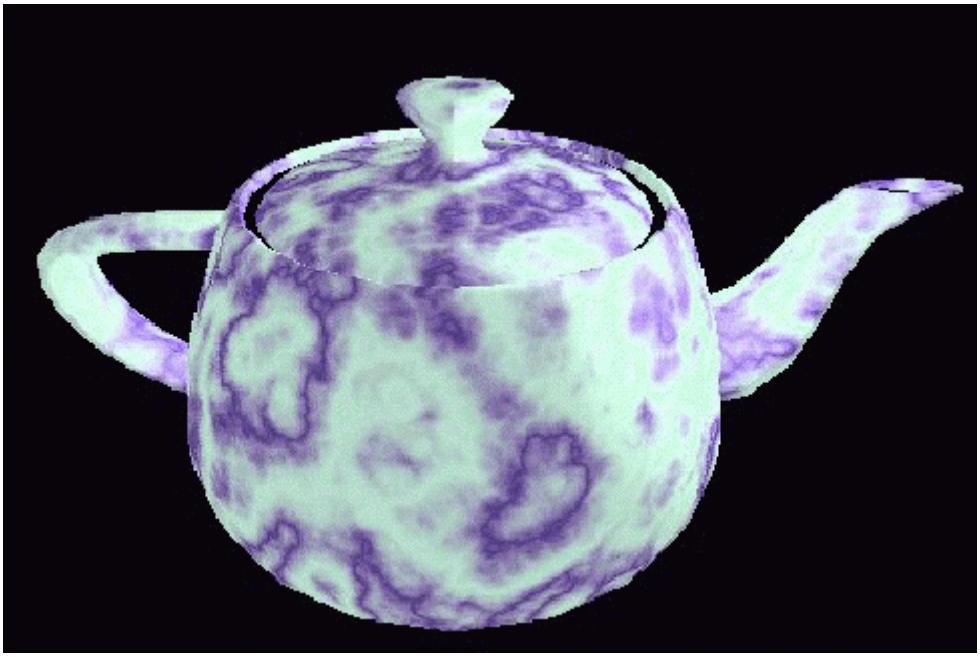
- minimaler Speicheraufwand
- die Texturwerte können an jeder Stelle mit optimaler Genauigkeit berechnet werden
- die Textur ist auf der gesamten Fläche definiert

Nachteile:

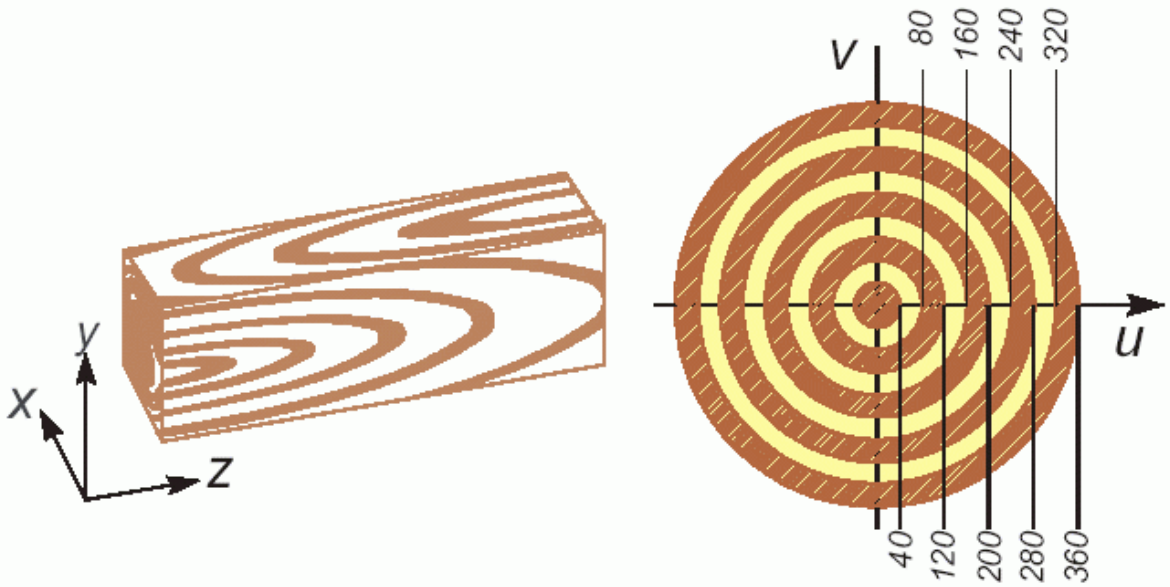
- große Vielfalt an unterschiedlichen Verfahren, Standardisierungen schwierig
- selbst Experten haben Probleme, komplexe Texturen, die sie bildlich vor Augen haben, in mathematische Formeln umzusetzen!

häufig: Kombination des prozeduralen Ansatzes mit *3D-Textur-Ansatz:*

- Texturfunktion wird nicht als $C(u, v)$ in 2 Variablen (flächig), sondern als $C(u, v, w)$ für Volumina definiert.
- Das zu texturierende Objekt wird von diesem Volumen umschlossen
- der Texturwert an einem Objektpunkt ergibt sich aus dem Funktionswert an der jeweiligen Stelle: Objektoberfläche schneidet Textur-Muster aus dem Volumen heraus
- Oft für Maserungen und Muster wie z.B. Holz, Marmor.



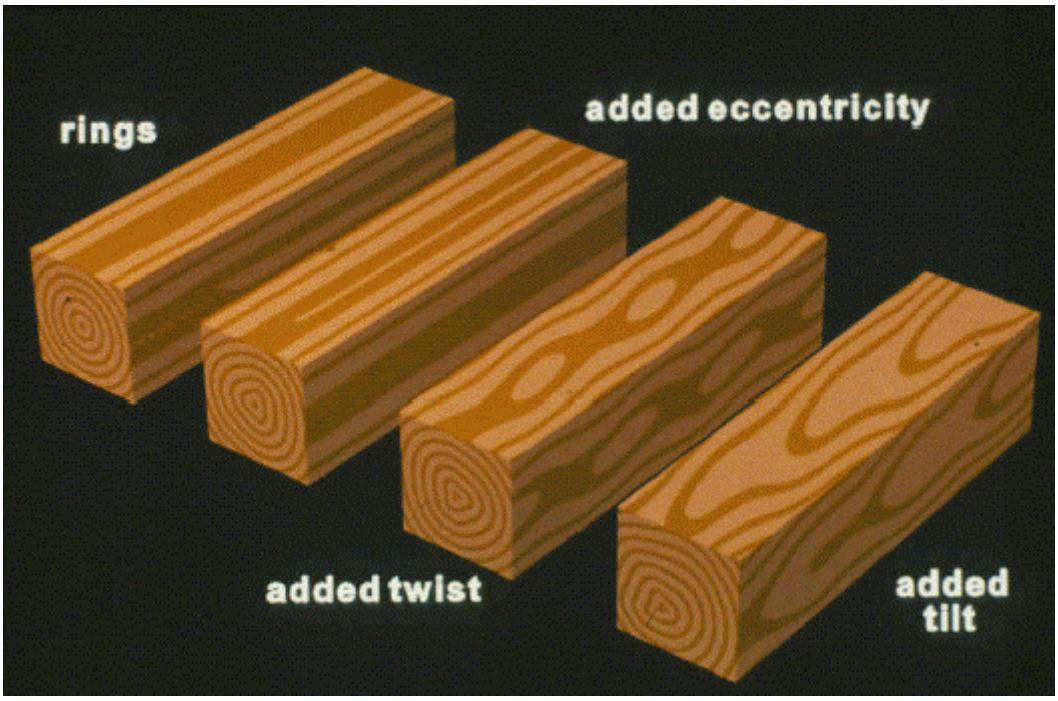
Beispiel: Erzeugung einer Holztextur mit Hilfe einer Prozedur



Beispiel: Erzeugung einer Holztextur mit Hilfe einer Prozedur (Pseudokode)

```
procedure wood(var x,y: real; var r,b,g: real);
var radius, angle: real; grain: integer;
begin
   radius := sqrt(x*x+y*y)
   grain:= round(radius) mod 80
   if grain < 40 then
        begin r:=r_light; g:=g_light; b:=b_light
        end else
        begin r:=r_dark; g:=g_ dark; b:=b_dark
        end
    end

(r,g,b) := wood(x,y)
```


rings
added eccentricity
added twist
added
tilt

Beispiele für komplexe optische Effekte durch Kombination mehrerer Texturen und Texturierungstechniken: