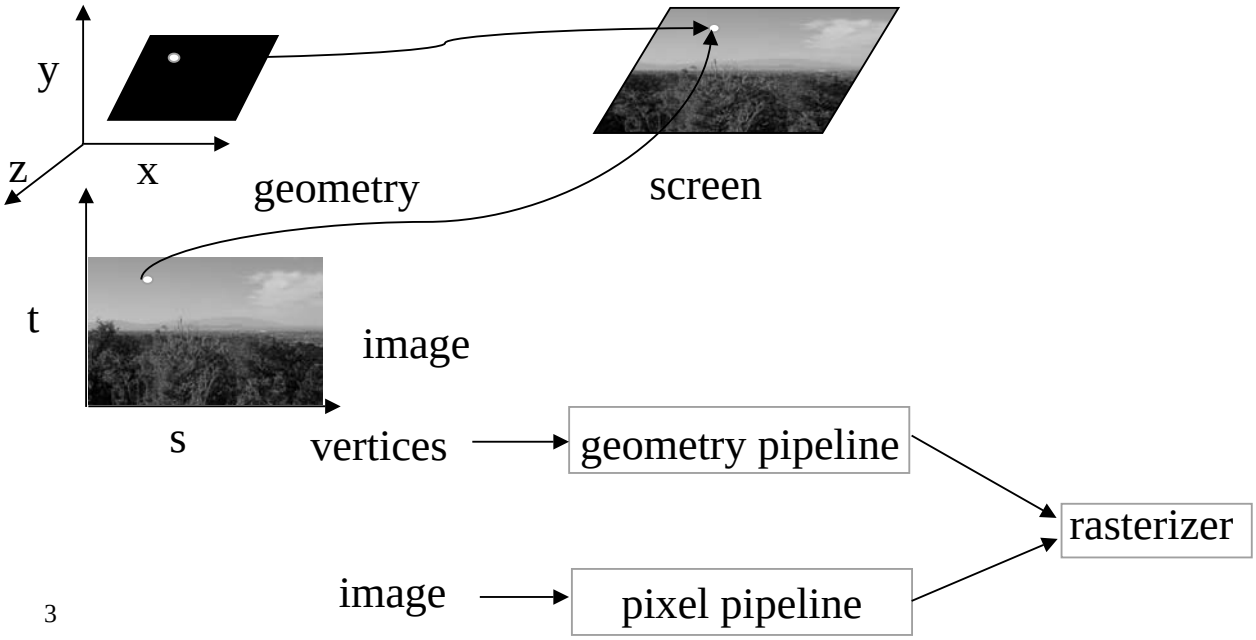
Kapitel 2: Texture Mapping

Überblick: Texture Mapping

- Einführung
- Texturquellen
- Abbildung, Parametrisierung, Texturkoordinaten
- Filterung
- Perspektivisch korrekte Interpolation
- Texturen in OpenGL
- Lightmaps
- Environment-Mapping
- Bump-Mapping
- Volumentexturen

Texture-Mapping

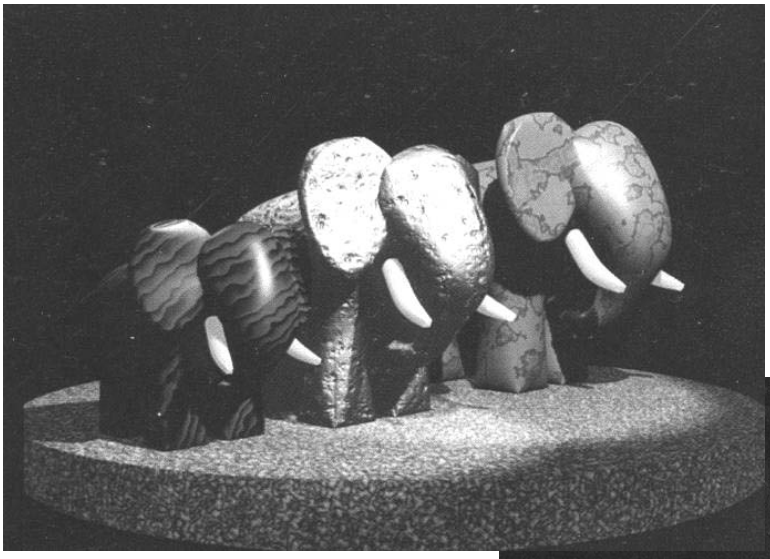
- **Realistischeres Aussehen durch Feinstrukturierung der Oberfläche pro Pixel**
- **Kombiniere Geometrie mit Bildern**



Texturen als eine von vielen Mapping-Arten

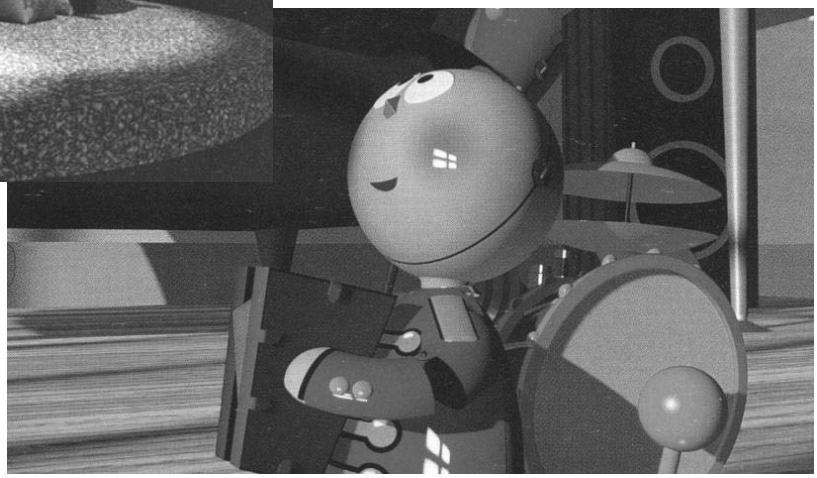
- **Reflexionseigenschaften**
  - Farbe, Reflexionskoeffizienten, Transparenz
  - klassisches „Texture-Mapping“
- **Beleuchtung**
  - „Environment-Mapping“, „Reflection-Mapping“
  - „Shadow-Mapping“, „Illumination-Mapping“
- **Geometrie**
  - verschiebe Flächen
  - „Displacement-Mapping“
- **Normalenvektoren**
  - $\underline{N}(\underline{P}) = N(\underline{P} + t\,\underline{N})$ oder $\underline{N} = \underline{N} + \underline{dN}$
  - „Bump-“ oder „Normal-Mapping“

Mapping: Beispiele



•Solid-Texture

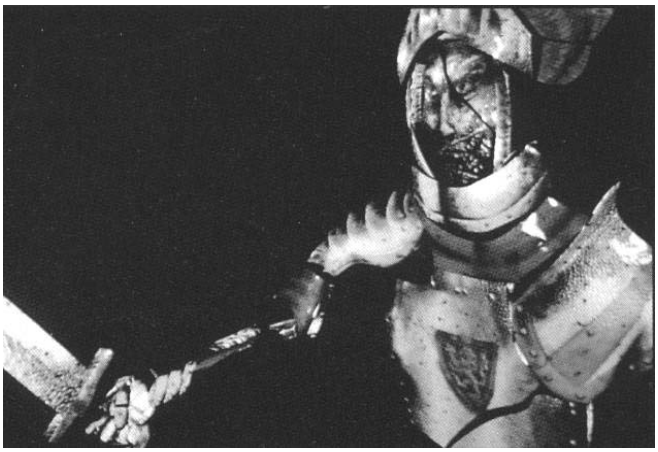
•Bump-Mapping

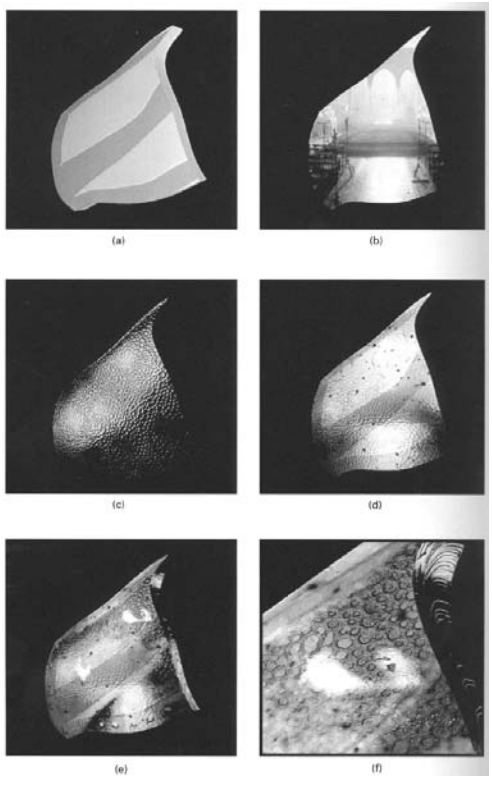


•Reflection-Mapping

Mapping: Beispiele

- **Modulation des Bildsyntheseprozess**
  - Farbe
  - Fläche
  - Normale
  - Beleuchtung





Textur-Quellen

- **Diskrete Texturen aus Bildern (Fotographien, Simulationen, Videos, ...)**
  - einfache Akquisition
  - Signalrekonstruktion notwendig (Interpolation)
  - beschränkte Auflösung, Aliasing
  - hoher Speicherbedarf
  - Vorsicht: „Eingefrorene Beleuchtung“
  - 1D Bilder (Farbtabellen) und 3D Bilder (Volumina)

Textur-Quellen

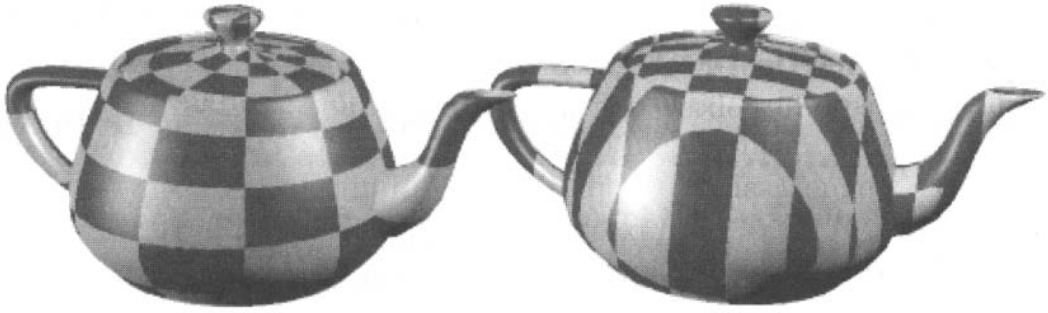
- **Prozedurale Texturen durch Algorithmen (Shader)**
  - nicht-triviale Programmierung
  - unbeschränkte Auflösung, optimale Genauigkeit
  - geringer Speicherbedarf
  - beliebige Algorithmen

Dimensionalität von Texturen

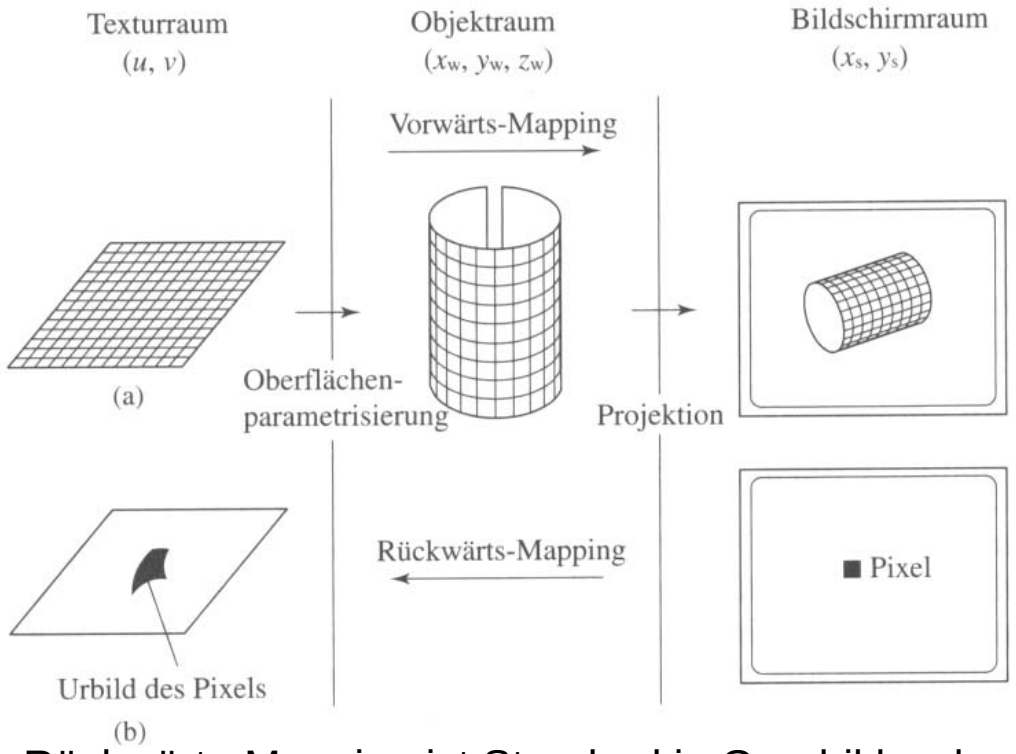
- **Textur: Diskret abgetastete Abbildung**
- **„nD-Textur“ definiert meist den Definitionsbereich:**
  - 1D: Parameter einer Linie, Höhe, Temperatur, ...
  - 2D: Parameter einer Oberfläche (u, v), Richtung (θ,φ), ...
  - 3D: Punkt im Raum (Solid-Texture), Richtungs-Vektor, ...
  - 4D: Linien im Raum (Reflexion), IBR
- **Bildraum (Wertebereich):**
  - 1D: Parameter für Reflexion, Intensität/Helligkeit, ...
  - 2D: Intensität und Transparenz, ...
  - 3D: Farbe, Normalenvektor, Reflexionsvektor, ...
  - 4D: Farbe/Vektor und Transparenz, ...

Texturabbildung

- **Bildschirmkoordinaten (x_s, y_s)**
- **Weltkoordinaten (x_w, y_w, z_w)**
- **Flächenparameter (u,v) (z.B. Bezier-Patch)**
- **Texturkoordinaten (s, t, r), manchmal auch (u,v)**
- **Richtungsvektor (R, N, H, …)**
- **Funktion der obigen Parameter (z.B. in Shadern)**
- **Projektion auf andere Flächen (Kugel, Zylinder, Box, ...)**

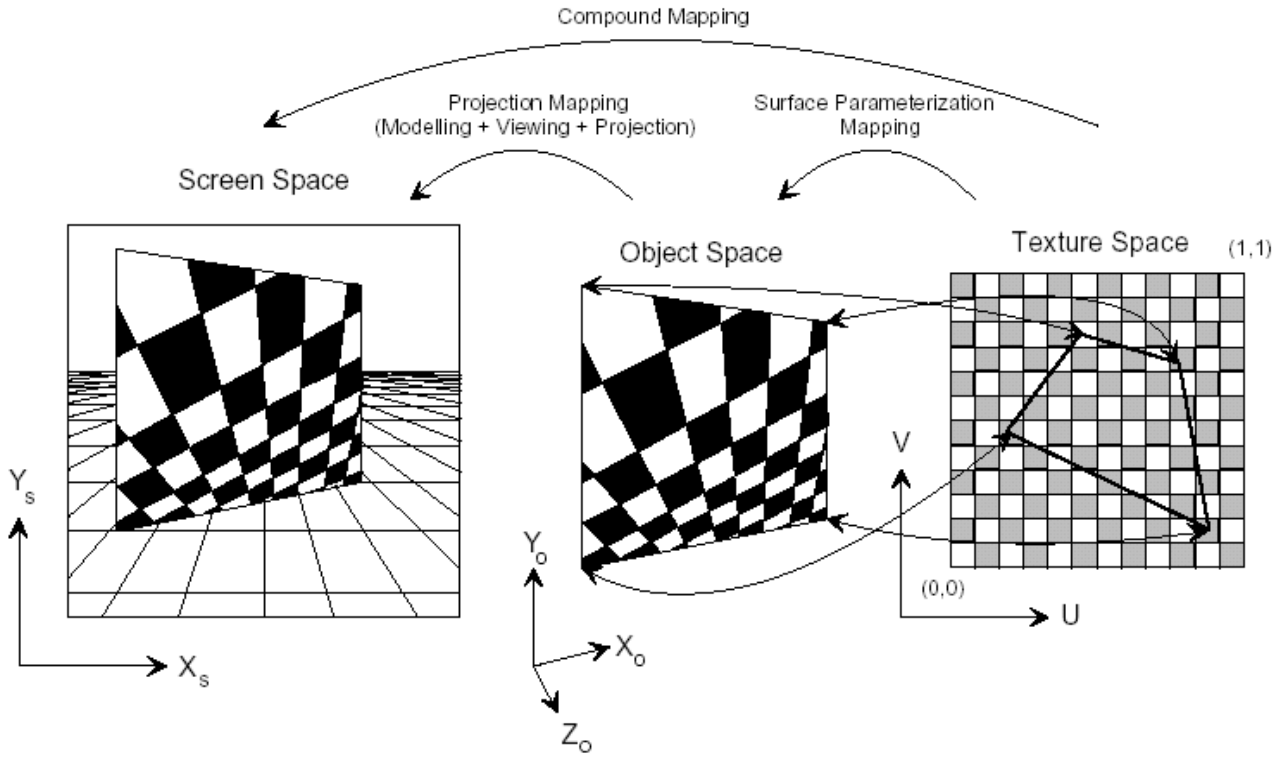


Vorwärts/Rückwärts-Mapping



Rückwärts-Mapping ist Standard in Graphikhardware

Zweistufiges Forward-Mapping



Vorwärts-/Rückwärts-Mapping

- **Ausgangspunkt:**
  Zusammengesetzte Abbildung (Compound Map) τ
- **Direkte Abbildung (texture scanning)**

```
for u
    for v
          compute (x_s,y_s)= τ(u,v)
          color(x_s,y_s) = texture(u,v)
```

- **Inverse Abbildung (screen scanning)**

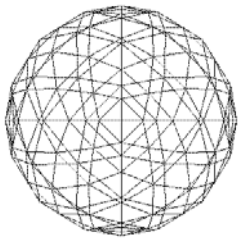
```
for x_s
    for y_s
          compute (u,v) = τ^-1(x_s,y_s)
          color(x_s,y_s) = resample(texture(u,v))
```

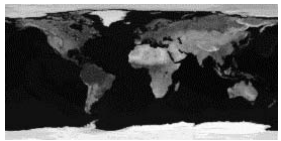

Standard-Parametrisierungen

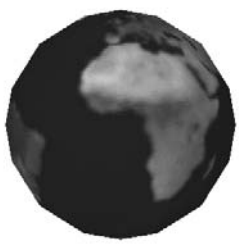
Parametrisierung im Objektraum

- Zylinderkoordinaten (ϕ,z)
- Polarkoordinaten (ϕ,θ)
- analog für Kegel, Torus, ...

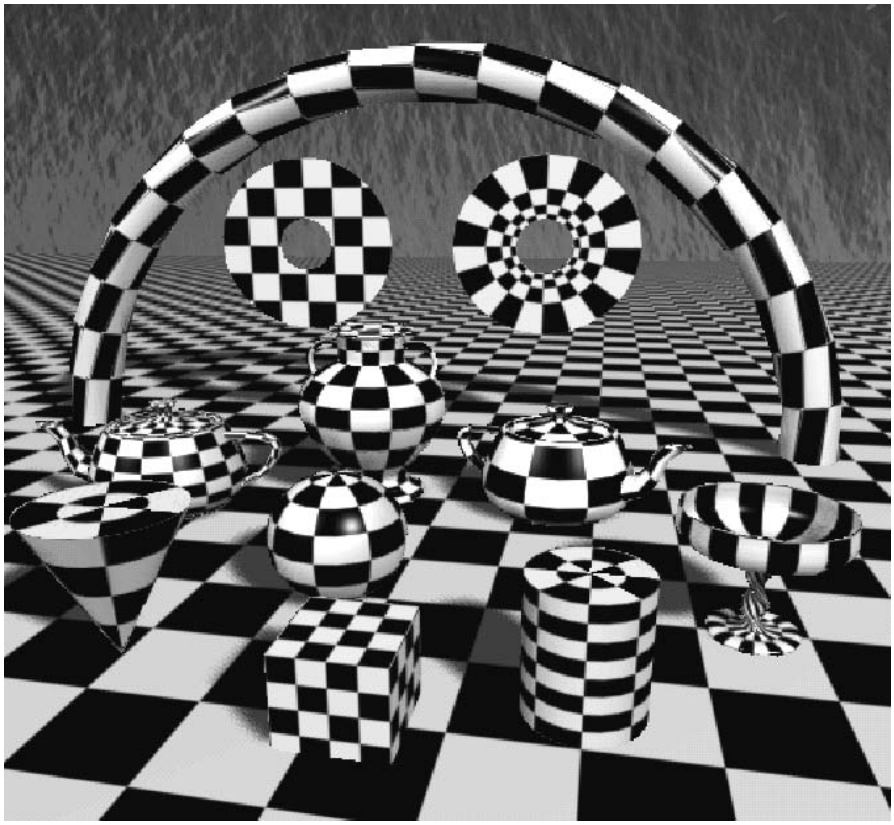
$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \phi / 2\pi \\ z / h \end{pmatrix}$$





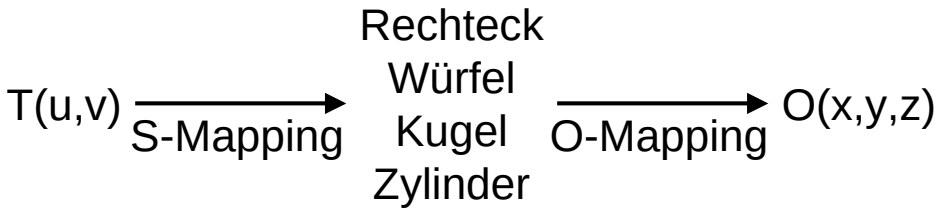


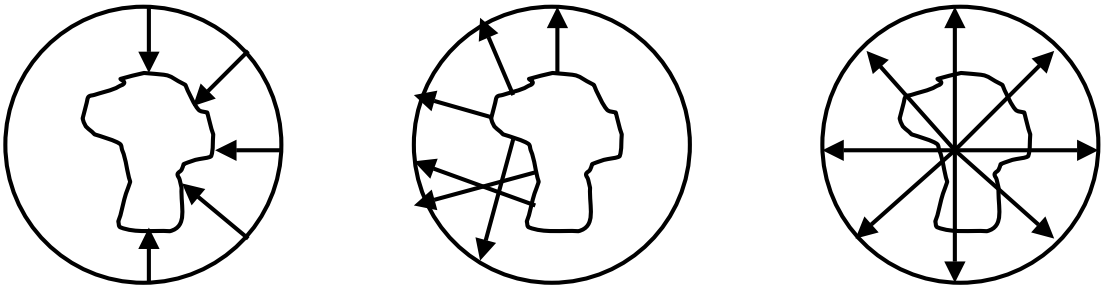
Standard-Parametrisierungen



Zweiphasen-Mapping

- **Verwende Standardkörper als Zwischenschritt**





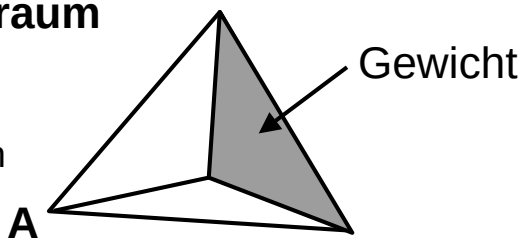
Normale der Hilfsfläche -> Oberfläche

Normale der Oberfläche -> Hilfsfläche

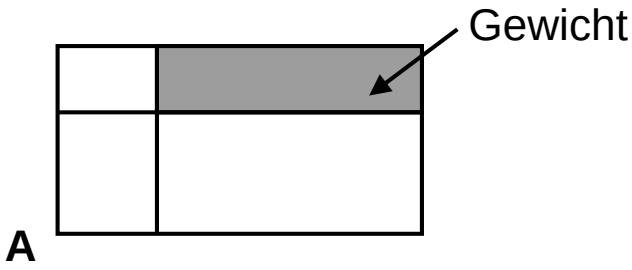
Linie durch Mittelpunkt

Parametrisierung von Polygonen

- **Parametrisierung im Objektraum**
  - Lineare Interpolation im Dreieck, über baryzentrische Koordinaten



  - Bilineare Interpolation im Viereck



- **Parametrisierung im Bildraum: inverse zusammengesetzte Abbildung τ^{-1}**

Parametrisierung von Polygonen

- **Bildverzerrung durch Projektion: rationale lineare Transformation (homogene Koordinaten)**
  - Vom Texturraum in Clip/NDC:

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \mathbf{M}_{TS} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \qquad \begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \end{pmatrix}$$

  - Umkehrung:

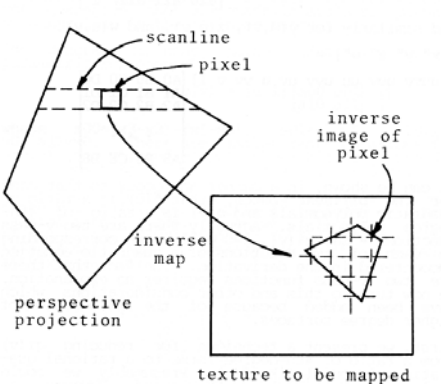
$$\begin{pmatrix} u' \\ v' \\ q' \end{pmatrix} = \mathbf{M}_{ST} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \quad \text{mit} \quad \mathbf{M}_{ST} = \mathbf{M}_{TS}^{-1} \qquad \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u'/q' \\ v'/q' \end{pmatrix}$$

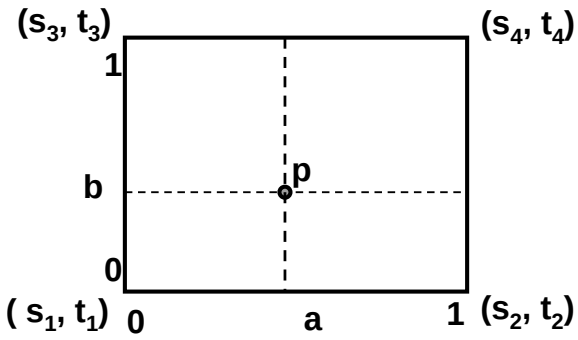
- **Einfacher als Umkehrabbildung:**
  - Scanline-Interpolation in der Bildschirmebene (lineare Interpolation in ***(u', v', q')***)
  - perspektivisch korrekte Interpolation

Texturkoordinaten

- **Texturparameter üblicherweise in [0,1]**
  - Texturkoordinaten an den Eckpunkten einer Fläche
    - Berechnung der Texturkoordinaten des Pixels
  - Benötigt Flächenparametrisierung im Inneren
    - Dreieck (lineare Scanline-Interpolation)
    - Rechteck (bilineare Interpolation)

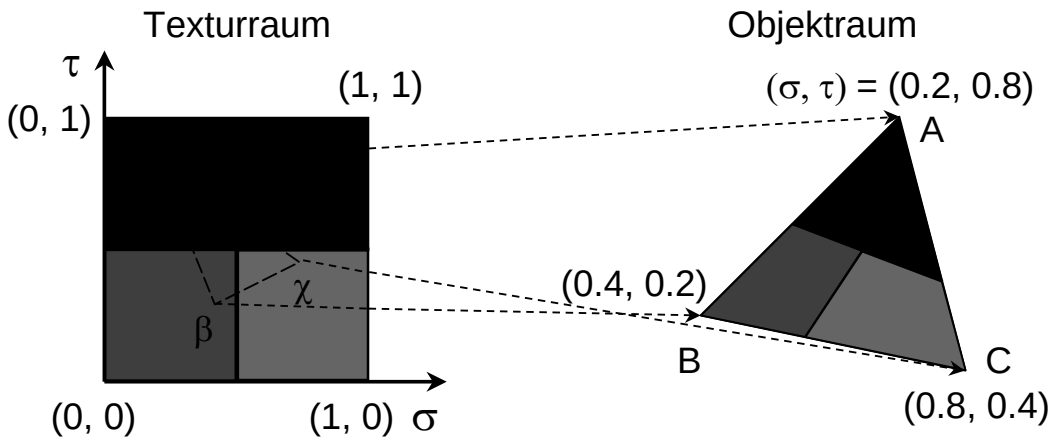
$$s = a*b*s_4 + (1-a)*b*s_3 + a*(1-b)*s_2 + (1-a)*(1-b)*s_1$$



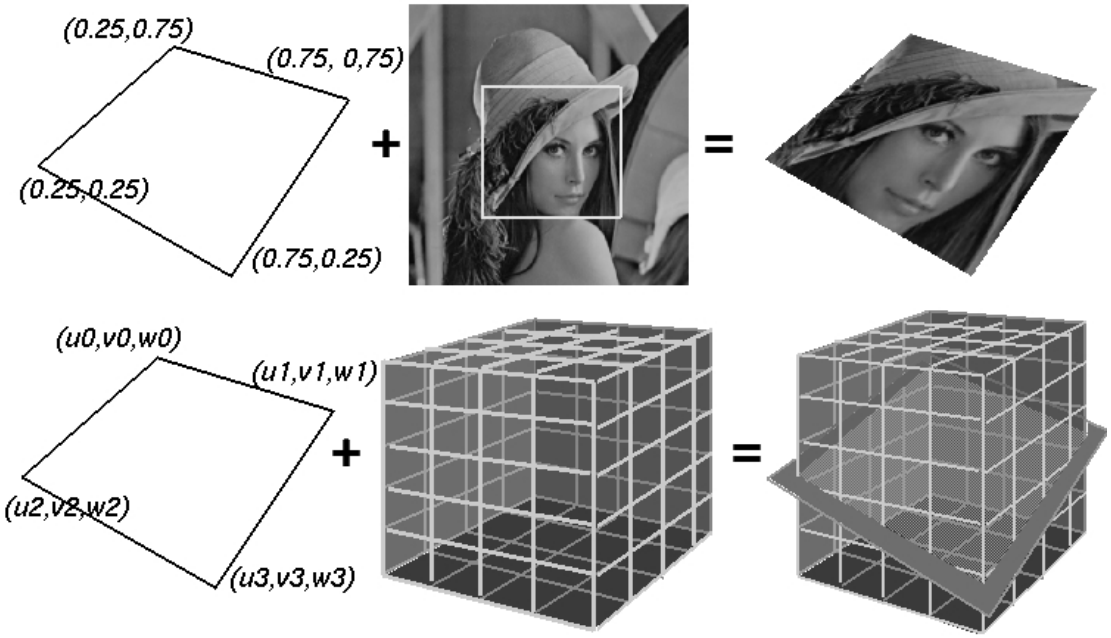


Texturkoordinaten

- **Texturkoordinaten an den Vertizes eines Dreiecks**



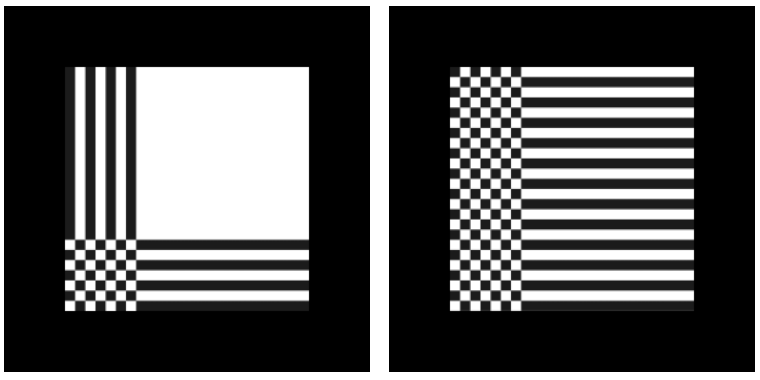
Texturkoordinaten in 2D und 3D



Texture Wrapping

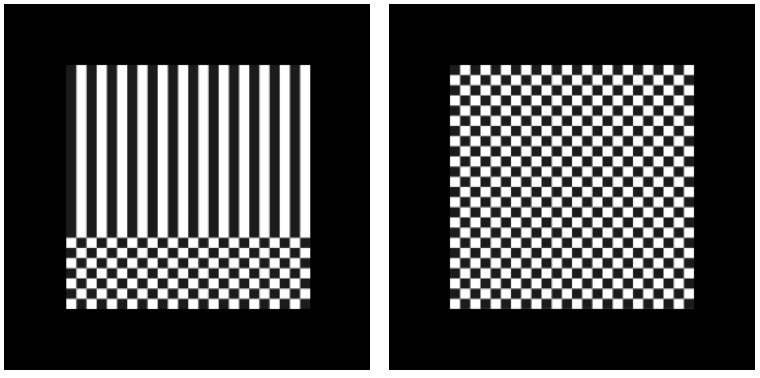
Wrapping: Fortsetzen einer Textur über [0,1] hinaus

clamp/
clamp



clamp/
repeat

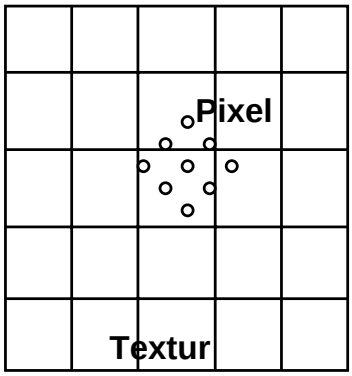
repeat/
clamp

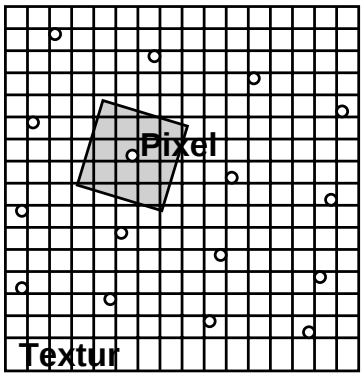


repeat/
repeat

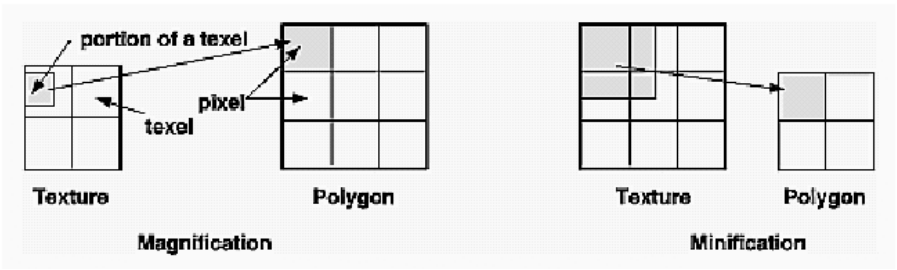
Textur-Filterung

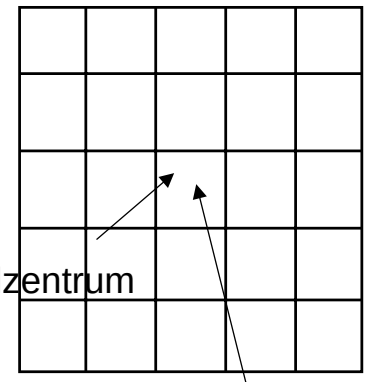
- Vergrößerung (Magnification)
  - Abbildung weniger Texel auf viele Pixel
  - Nearest Neighbor:
    - nächstliegendes Texel
  - Bilineare Interpolation:
    - Interpolation der 4 nächsten Texel
    - Nutzung der Nachkommastellen
    - Glättung
- Verkleinerung (Minification)
  - Abbildung vieler Texel auf ein Pixel
    - Aliasing:
      - Abtastung hochfrequenter Signale mit niedriger Frequenz
  - Filterung
    - Mittelung über (viele) zugehörige Texel
    - Im Allgemeinen zu teuer zur Laufzeit

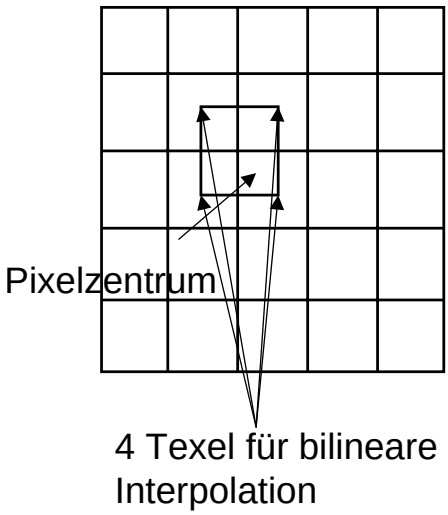




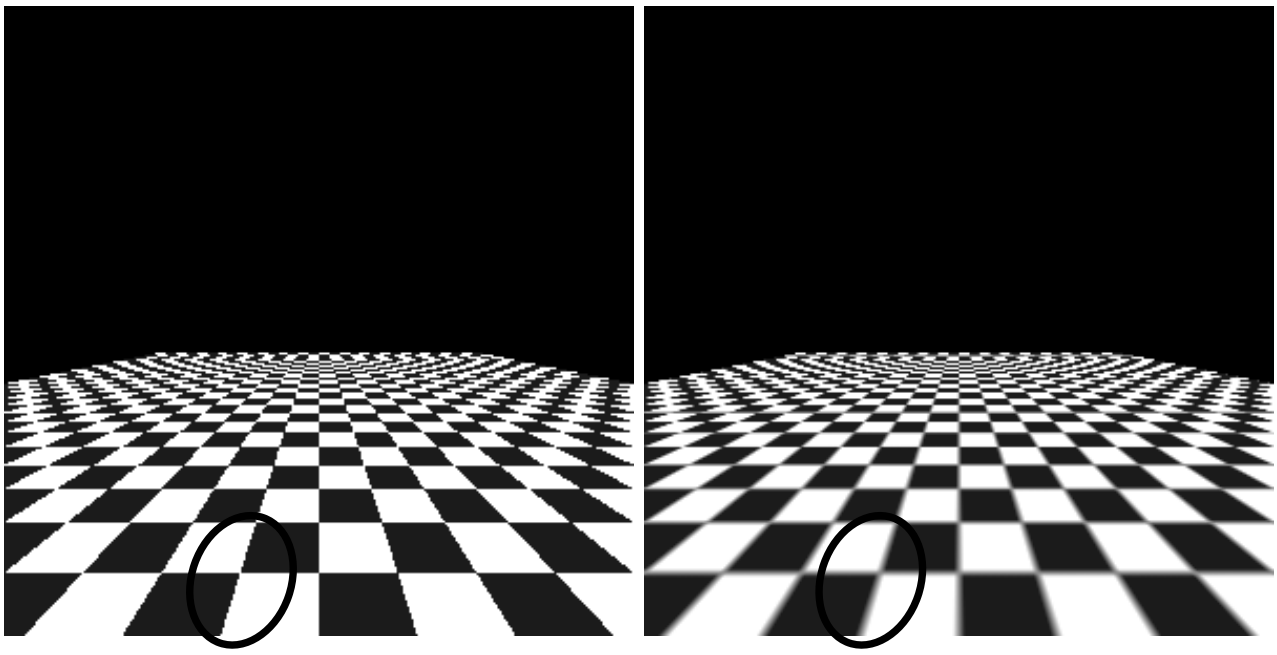
Minification/Magnification







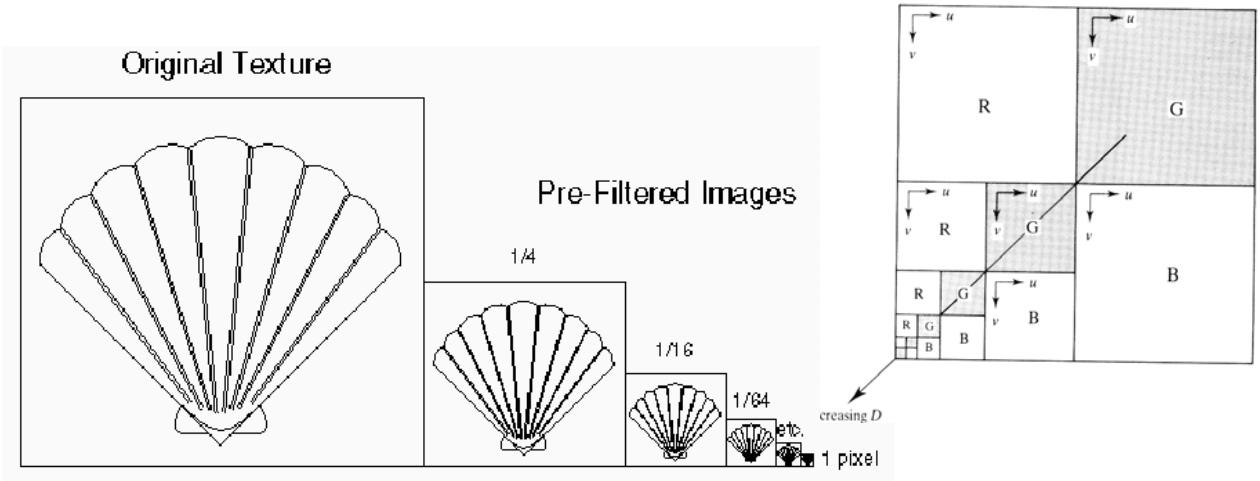
Magnification



Nearest Neighbor linear

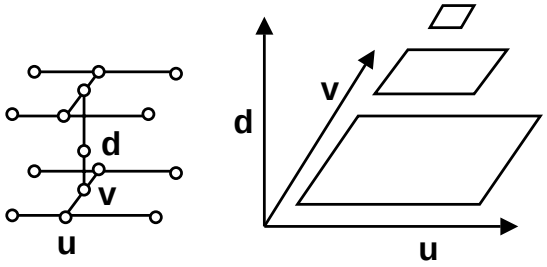
MipMapping

- **Einfache Vorfilterung von Texturen**
  - Multum-In-Parvo (MIP): Vieles im Kleinen
  - Speichere rekursiv Texturen halber Größe (1/3 Speicher mehr)
  - Filterung/Mittelung über je 2x2 Texel

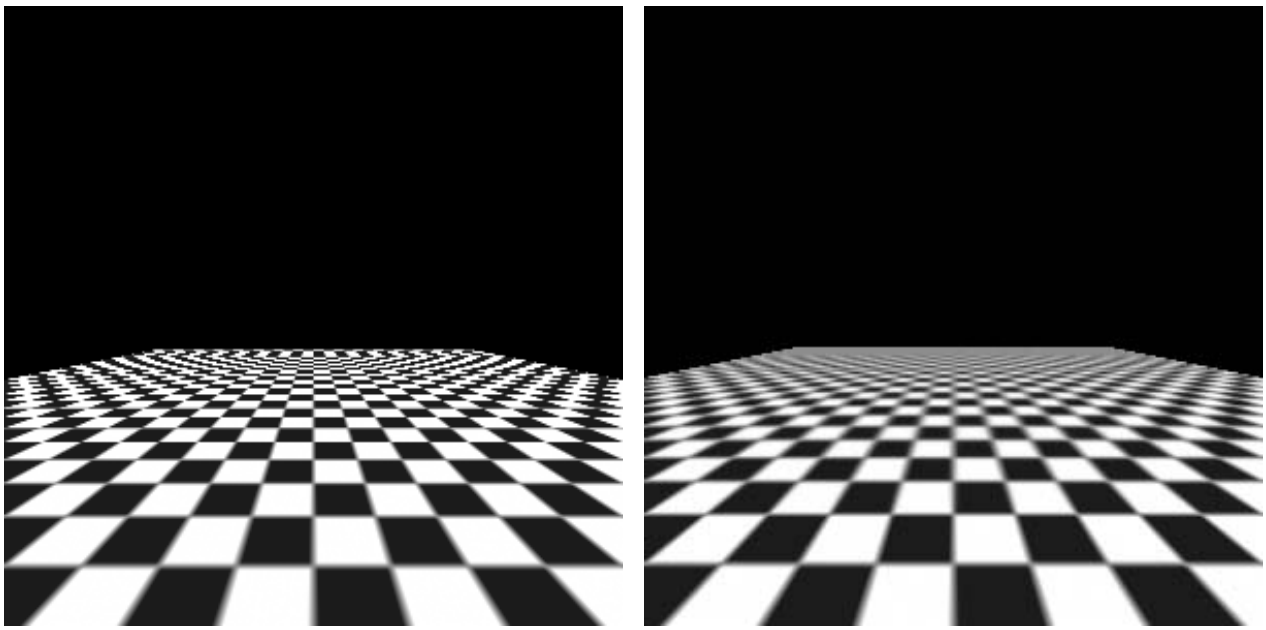


MipMapping *cont.*

- **Wähle Texturauflösung so, dass**
  - Texelgröße(n) ≤ Ausdehnung des Pixel auf Textur < Texelgröße(n+1)
- **Rekonstruktion**
  - Trilineare Interpolation der 8 nächstliegenden Texel
- **Problem**
  - Starkes „Blurring“ bei oft gefilterten MipMap Levels



MipMapping *cont.*

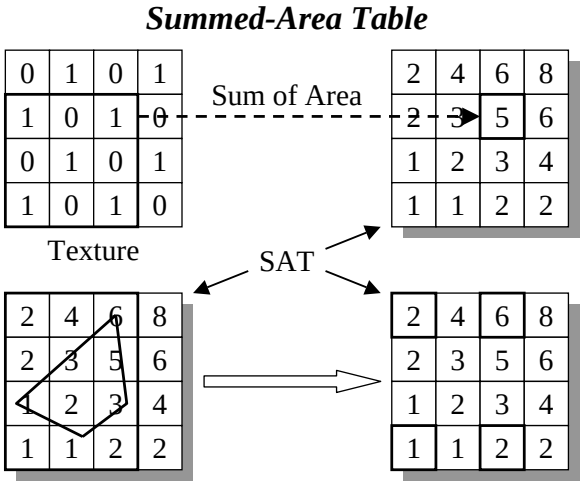


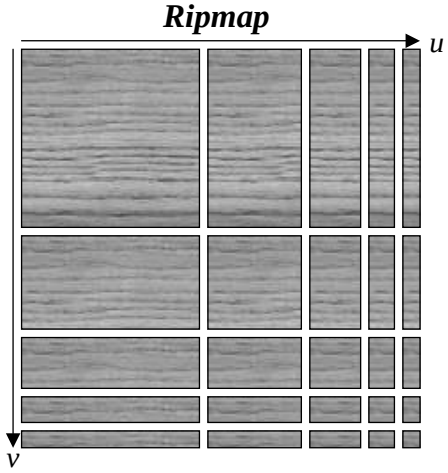
Ohne MipMapping

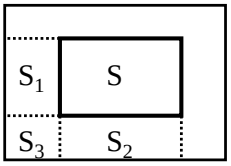
Mit MipMapping

Anisotrope Texturfilterung

- **Filtere nicht in alle Richtungen gleich**
  - Kein quadratischer Bereich im Texturraum
  - Typischerweise eine höher Anzahl von Samples (z.B. 8x8)
- **RIPmaps (rectengular MipMaps)**
- **Summed Area Tables (SAT)**



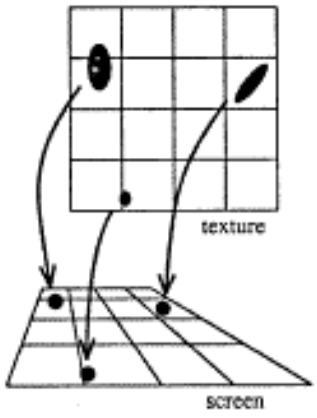


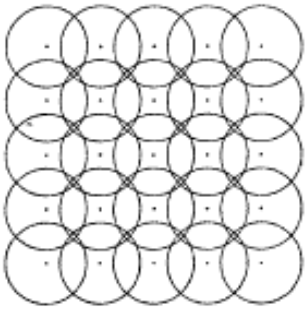


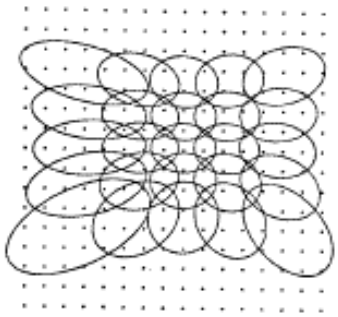
$S=S_{tot}- S_1 - S_2 + S_3$

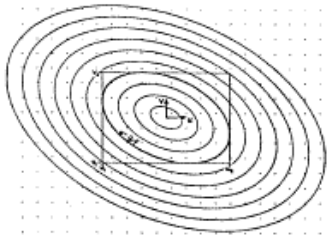
EWA Filterung

- **Elliptical Weighted Average (EWA)**
- **Kreisförmige, überlappende Pixel**
- **Gewichtetes Mittel aus konzentrischen Ellipsen**
- **Kreis in Bildschirmkoordinaten wird zu Ellipse im Texturraum**
- **Effiziente Berechnung im Texturraum**



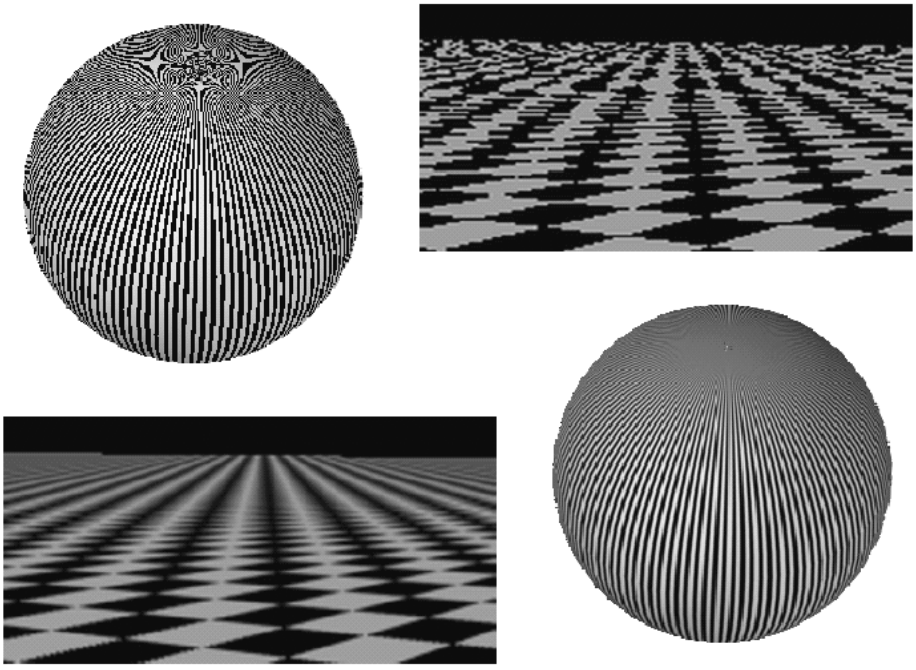




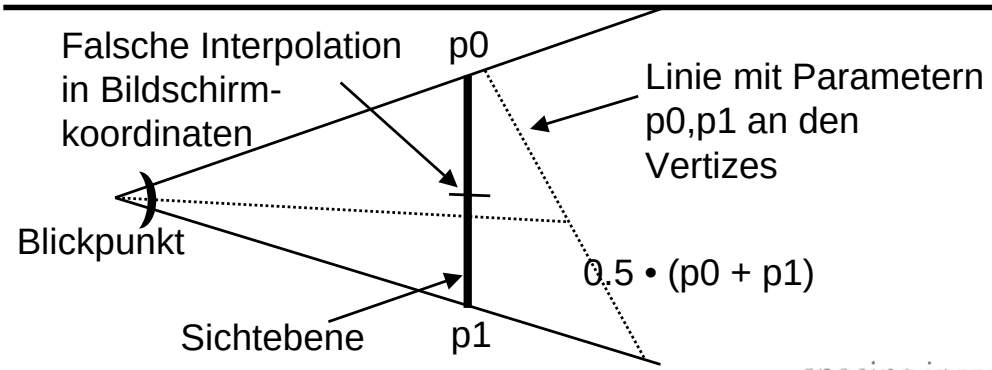


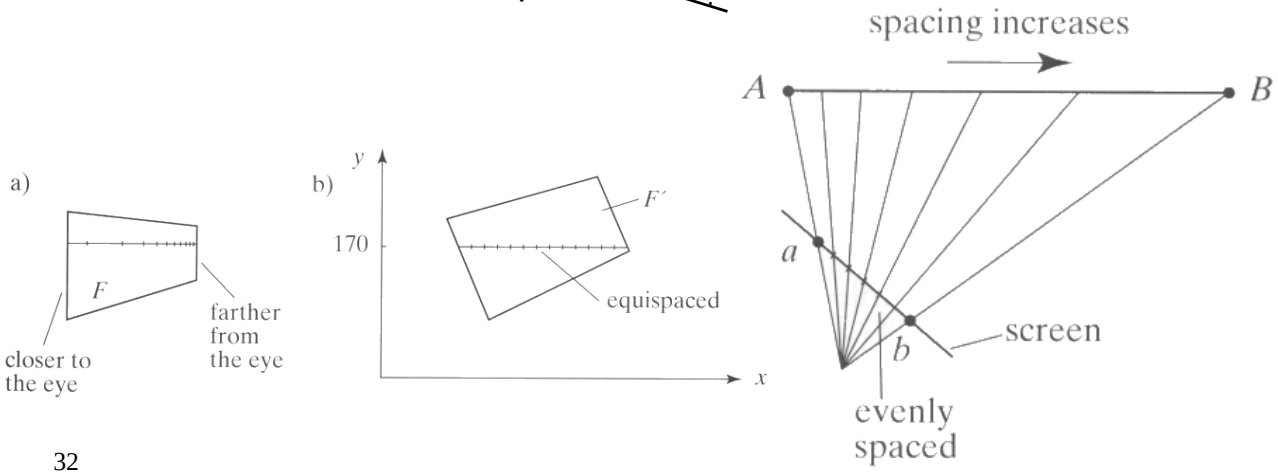
EWA Filterung *cont.*

- **Nicht direkt in Hardware unterstützt**
- **Beispiel:**



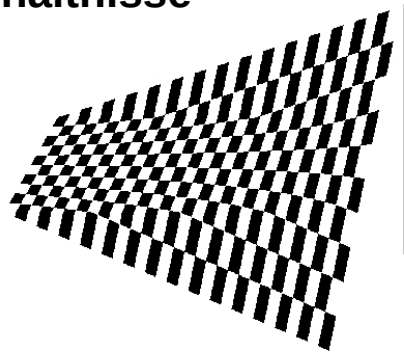
Texturinterpolation

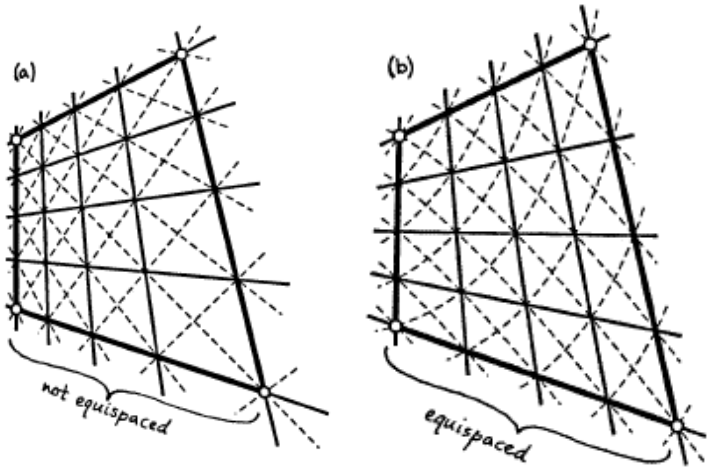


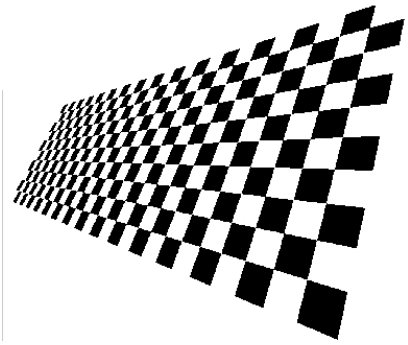


Perspektivisch korrekte Interpolation

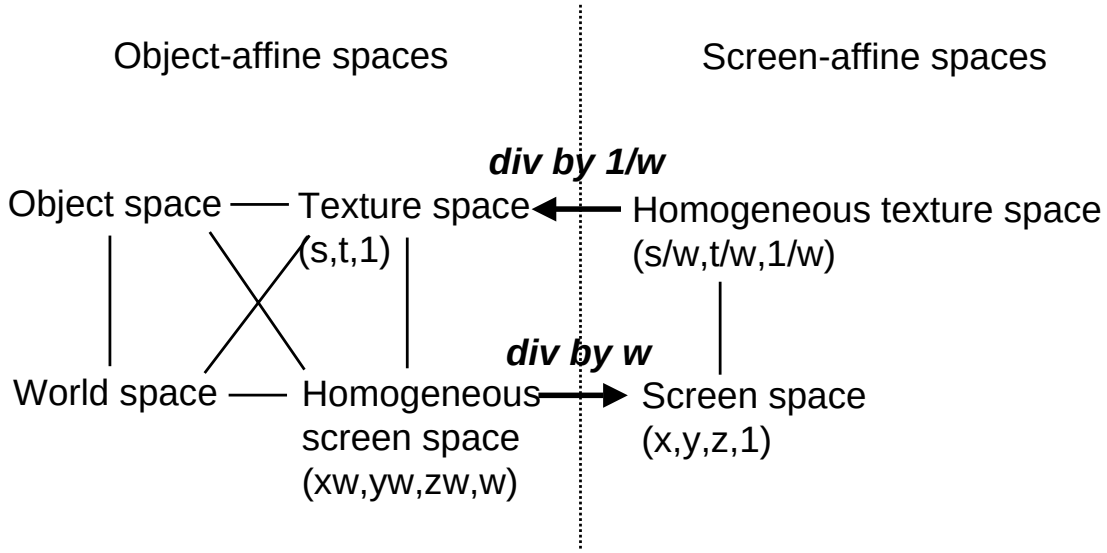
- **Nichtlineare z-Transformation führt zu Verzerrungen**
- **Lineare Interpolation erhält Teilverhältnisse**
- **Projektion erhält diese nicht**
- **Benötigt hyperbolische (rationale) Interpolation**







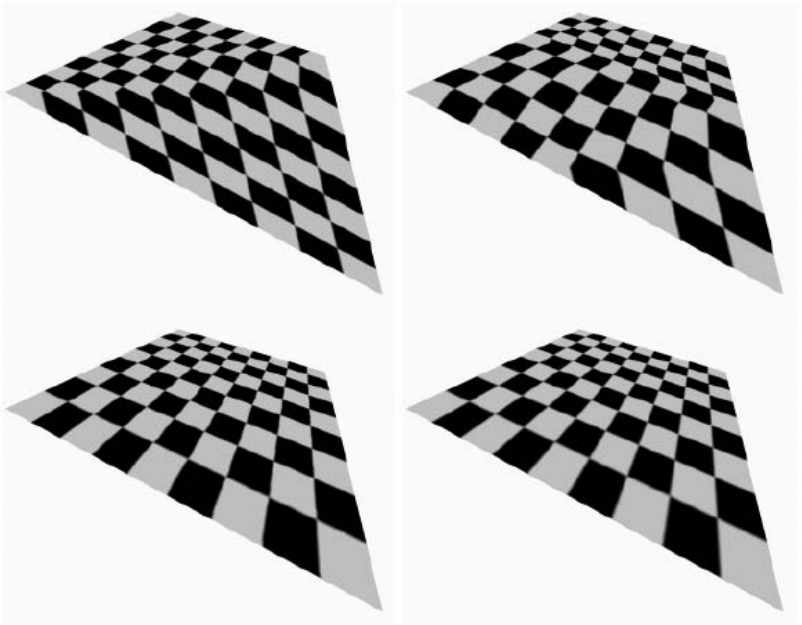
Perspektivisch korrekte Interpolation *cont.*



- Interpoliere linear s/w, t/w **und** 1/w
- Dividiere pro Pixel s/w und t/w durch 1/w

Texturinterpolation ohne Korrektur

- **Unterteilung in 1,4,64 Quads (= 2,8,128 Tris)**



Texturen in OpenGL

- **`glEnable(GL_TEXTURE_2D)`**
- **Textur definieren**
  ```
  glTexImage2D( target, level, int_format,
     w, h, border, format, type, *texels )
  ```
- **wie bei Images:** ***`format, type, w (2^m), h (2^n)`***
- **evtl. exotische interne Texturformate:**
  `GL_LUMINANCE12_ALPHA4`
  `GL_R3_G3_B2`
  `GL_RGBA12`
- **wenn nicht MipMap:** ***`level=0`***
- **Target:**
  `GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D`
  `GL_PROXY_TEXTURE_2D`

Texturquellen

- **Skaliere Bilder, die nicht $2^m x 2^n$**

```
gluScaleImage(.,w_in, h_in,..,w_out, h_out,..)
```

- **Modifiziere einen Teil einer definierten Textur**
  - evtl. viel schneller als **`glTexImage2D`**
  - beliebige Breite und Höhe (z.B. Video-Textur)

```
glTexSubImage*D(.,w, h, x_offset, y_offset,..)
```

- **Verwende Bild im Framebuffer als Textur**

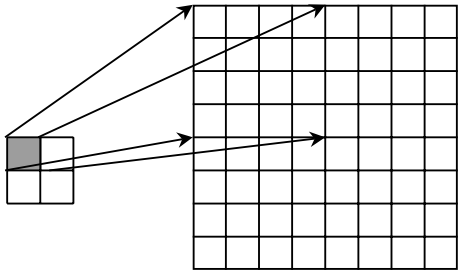
```
glCopyTexImage*D(...)
glCopyTexSubImage*D(...)
```

- **Neuere Entwicklungen:**
  - direktes Rendern in Textur
  - Rechtecke (texture rectangle)

Minification/Magnification

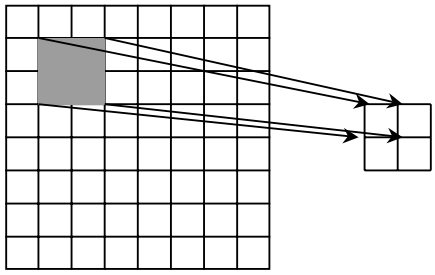
- **Textur-Filterung: Nearest Neighbor, (Bi)Linear**

```
glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_MAG_FILTER, GL_NEAREST)
glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_MIN_FILTER, GL_LINEAR)
```



Texture Polygon

Magnification



Texture Polygon

Minification

Mipmapping und korrekte Interpolation

- **Baue (und definiere) MipMap mit**

```
gluBuild2DMipmaps()
```

- **Spezifiziere Minification-Filter**

```
glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_MIN_FILTER,
  GL_NEAREST_MIPMAP_NEAREST)
glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_MIN_FILTER,
  GL_LINEAR_MIPMAP_LINEAR)
```

- **Empfehle perspektivisch korrekte Interpolation**

```
glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint)
```

  - **GL_DONT_CARE**
  - **GL_NICEST**
  - **GL_FASTEST**

Textur-Filter-Erweiterungen

- **Sharpen Texture Extension (SGI)**
  - Bilineare Interpolation im Mag-Filter schlecht für sehr nahe Texturen (z.B. Schrift)
  - Extrapolation einer höher aufgelösten Textur

  LOD(+1) = LOD (0) +1/4(LOD(0) – LOD(-1))

  2x2 Texelblöcke Org.Textur Kanteninformation 1.Mipmap-Stufe
- **Detail Texture Extension (SGI)**
  - für große unstrukturierte Texturen (Grass, Straße) ohne Kanten
  - Trenne 2k x 2k Textur in
    - 512^2 gefilterte Textur (tiefe Frequenzen)
    - 256^2 Textur-Ausschnitt (nur hohe Frequenzen)
  - und mische mit Blending

Textur-Funktionen

- **Wie wird die Textur-Farbe auf das Pixel angewandt?**
  - **Replace**: ersetzt Frag.-Farbe durch Texel-Farbe
    $C = C_t$, $A = A_t$
  - **Decal**: Alpha-Blending Frag.-Farbe mit Texel-Farbe
    $C = C_f(1-A_t) + C_tA_t$, $A = A_f$
  - **Modulate**: skaliert Texel-Farbe (mit Beleuchtung)
    $C = C_fC_t$, $A = A_fA_t$
  - **Blend**: interpoliert zwischen Frag.- und Env.-Farbe
    $C = C_f(1-C_t) + C_cC_t$, $A = A_f A_t$

```
glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
  GL_BLEND)
glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR,
  blend_color)
```


Texture Wrapping und Tiling

- **Textur-Verhalten außerhalb [0,1]**

```
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D,
     GL_TEXTURE_WRAP_T, GL_REPEAT )
```

- **Tiling, wenn größte Textur zu klein: spezifiziere Texturrand (border) für korrekte Interpolation**

Texturobjekte

- **Aktive Textur ist Teil des OpenGL-Zustands**
- **Häufiger Texturwechsel aufwändig**
  - Konvertierung der Textur in internes Format
  - Laden der Textur in den Texturspeicher bzw. in OpenGL-Server
- **vor OpenGL 1.1**
  - packe Textur-Definitionen in Display-Listen
  - OpenGL-Server erkennt diese und verwaltet Texturspeicher
- **seit OpenGL 1.1**
  - Texturobjekte (texture objects): vordefinierte Texturen mit Namen
  - einfaches Umschalten der Bindung
  - mehrere Texturen im Texturspeicher

```
glGenTextures(n, *texIDs)
glBindTexture(target, id)
```


Verwaltung des Texturspeichers

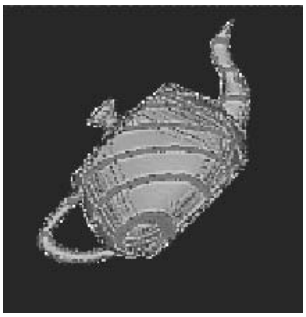
- **Caching von Texturobjekten beeinflussen**
  - **`glDeleteTextures()`**
  - **`glAreTexturesResident()`**
  - **`glPrioritizeTextures()`** Priorität [0,1]
- **Texture-Proxy**
  - Passt die Textur in den Speicher?
  - **`glGetIntegerv(GL_MAX_TEXTURE_SIZE,...)`**
  - Verwende **`GL_PROXY_TEXTURE_2D`** als Target
  - **`glGetTexLevelParameter*()`** sagt, ob es passt

Automatisch erzeugte Texturkoordinaten

- **Explizit setzen mit `glTexCoord()`**
- **Automatisch erzeugen mit `glTexGen{ifd}[v]()`**
- **spezifiziere eine Ebene** $Ax + By + Cz + D = 0$
  - Texturkoordinaten werden entsprechend dem Abstand des Vertex von der Ebene gesetzt
- **Arten:**
  - **`GL_OBJECT_LINEAR`**
    in Objektkoordinaten: Textur fest auf Objekt
  - **`GL_EYE_LINEAR`**
    in Augkoordinaten: Textur dynamisch auf bewegtem Objekt
  - **`GL_SPHERE_MAP`**
    Environment Mapping

Automatische Generierung - Beispiele

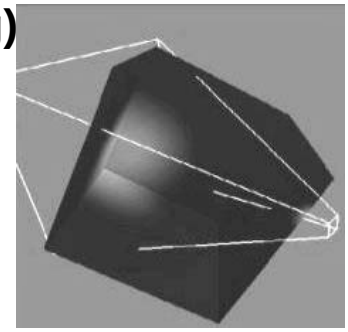
- **Konturen im Abstand von einer Ebene**
  1D-Textur Schwarz-Weiß mit Wrap

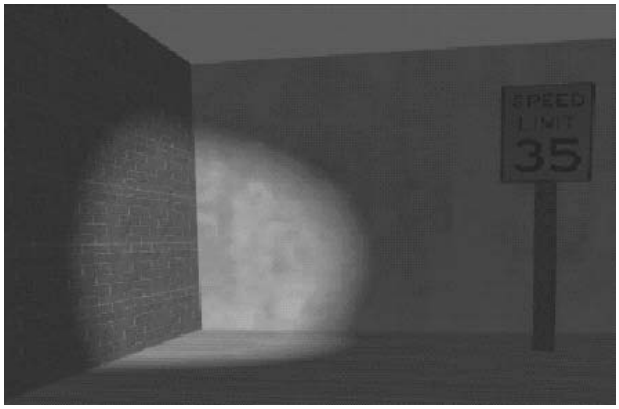




- **Höhenlinien in Terrain**
- **Erzeugung von 3D Texturkoordinaten, die den räumlichen Koordinaten des Vertex entsprechen (in Augkoordinaten)**
  Abstand von yz, xz, xy-Ebene

Texturmatrix

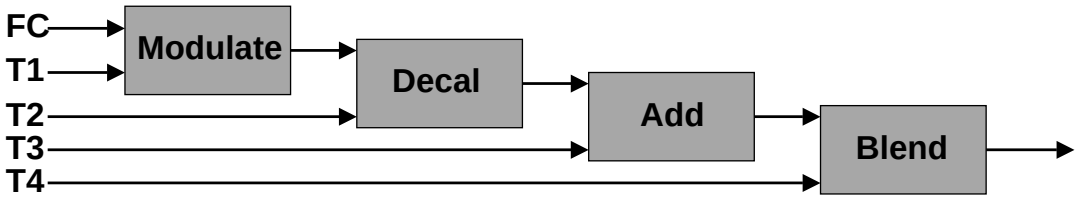
- **Texturkoordinaten können vor der Anwendung (evtl. nach automatischer Generierung) skaliert, rotiert, projiziert werden**
- **4 x 4 Matrix für (s, t, r, q) auf extra Stack `glMatrixMode(GL_TEXTURE)`**
- **Typische Anwendung: Texturprojektion**
  - Diaprojektion
  - Spotlights
  - generiere Texturkoordinaten wie Augkoordinaten (s,t,r,q)
  - definiere Modelview und Frustum wie üblich
  - skaliere/verschiebe NDC ->[0,1]





Multi-Texturen

- **Statt Multi-Pass Rendering komplexe Pixel-Operationen in einem Schritt (single pass)**
- **Blending mehrerer Texture Lookups**

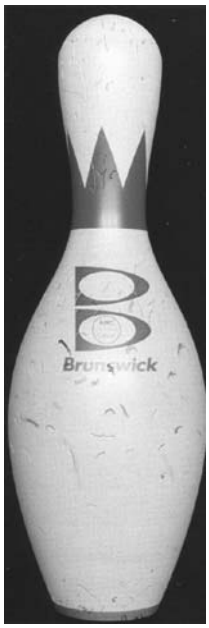


- **Jede Textur-Einheit hat eigenen Zustand (Image, Filtering, Environment, Matrix, ...)**
- **ARB_multitexture Erweiterung von OpenGL**

```
glActiveTextureARB(GL_TEXTUREn_ARB)
glMultiTexCoord4f(GL_TEXTURE0_ARB,
  s0, t0, r0, q0)
```

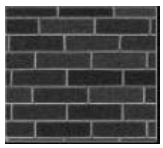
Multi-Texturen *cont.*

- **Komplexe optische Effekte durch Kombinationen mehrerer Texturen**

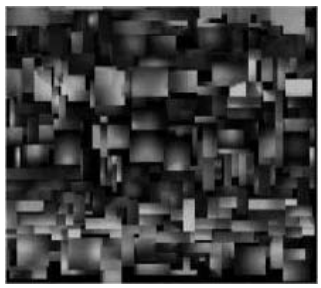


Lightmaps

- **Kombiniere Strukturtextur mit Lichttextur**
- **Wiederverwendung von Material bei unterschiedlicher Beleuchtung**
- **Lightmaps für diffuses Licht**
  - nur Luminanz-Kanal
  - geringe Auflösung (diffus)
  - können gepackt werden



Reflectance Irradiance Radiosity



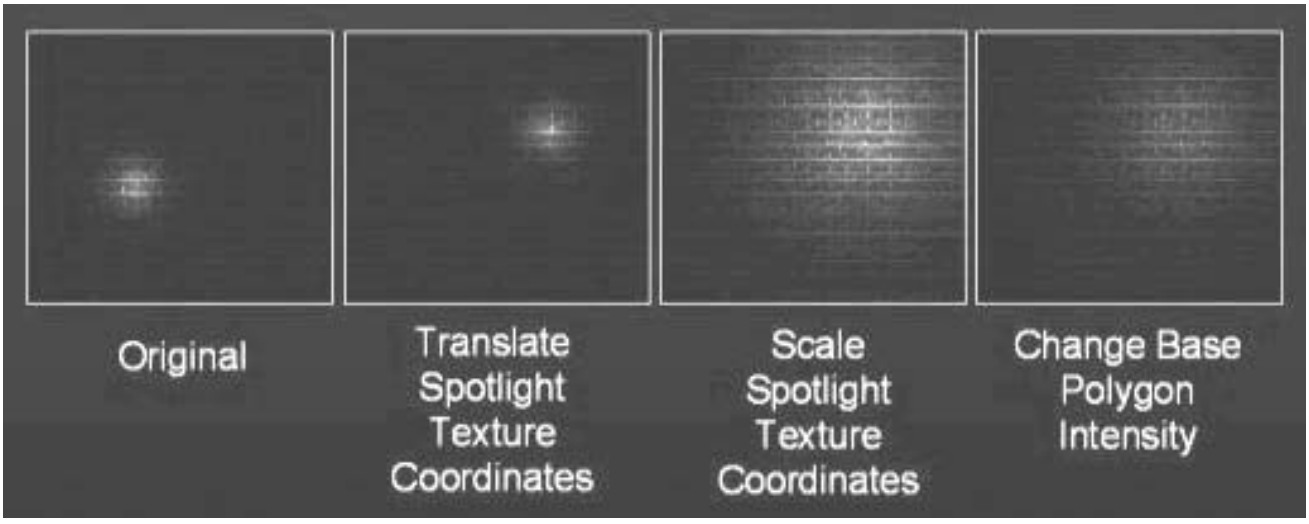
Quake2 Lightmap

Lightmaps - Lichtquellen

- **Kombiniere vorberechnete Beleuchtung mit texturierter Szene**

Lightmaps

- **Veränderliches Spotlight durch Transformation der Texturkoordinaten mit Texturmatrix**



Phong-Shading mit Highlight-Texturen

- **Modifiziere Texturkoordinaten der Highlight-Textur entsprechend der Lichtquellenrichtung**

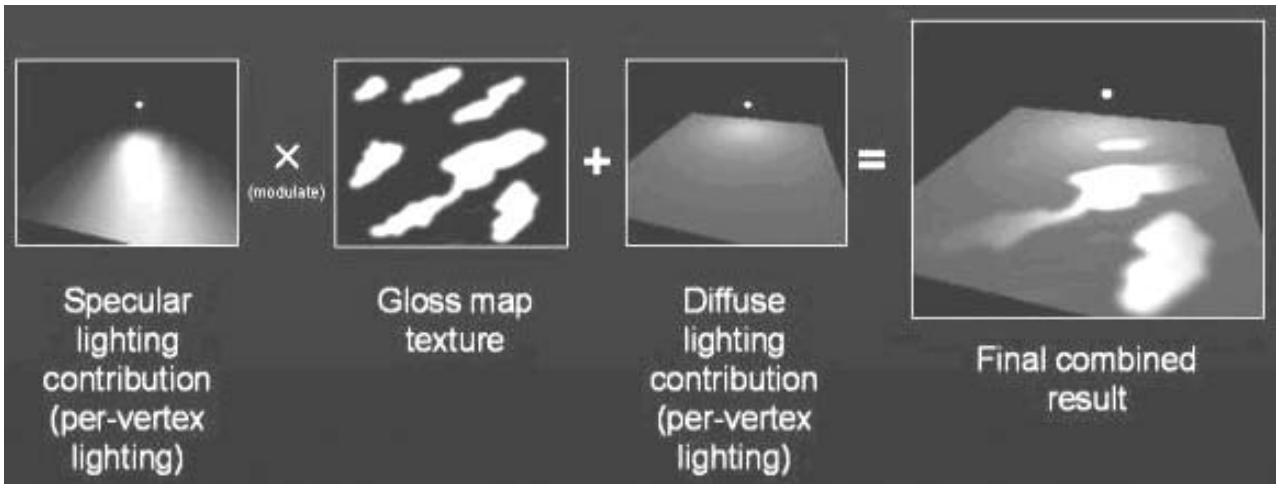
Texturiert mit diffuser Beleuchtung

Weißes Objekt mit Highlight-Textur moduliert

Überblendung

Gloss Maps

- **Lightmap für spekulare Reflexion**
- **Beispiel: spiegelnde Pfützen**



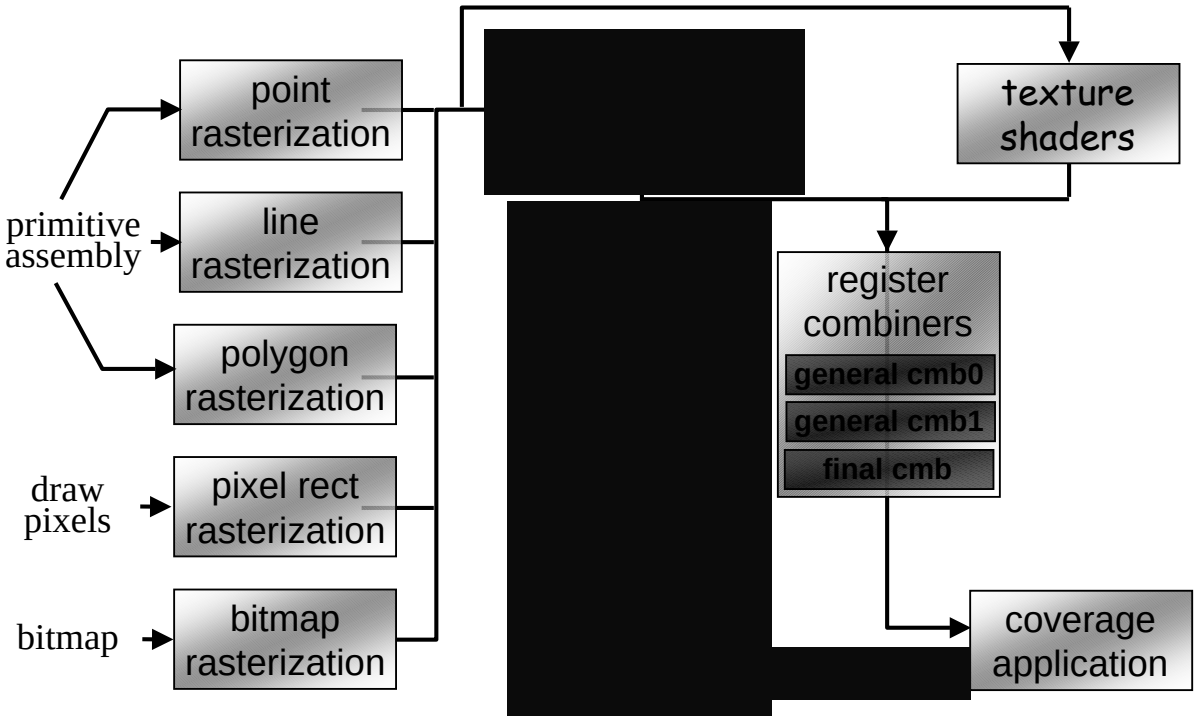
Pixel-Texturen

- **Für jedes Fragment Textur-Koordinaten nicht interpolieren, sondern frei bestimmen**
- **Interpretiere RGB-Werte im Framebuffer als Textur-Koordinaten (s, t, r)**
- **Beim Kopieren des Framebuffers auf sich selbst wird jedes Fragment entsprechend texturiert**
- **Farbmatrix beim Pixel-Transfer wirkt auf (s,t,r)**
- **Früher: Spezielle SGI Erweiterung bei Octane**
- **Heute: Dependent Texture (indirekte Adressierung) – kein Kopieren mehr nötig!**

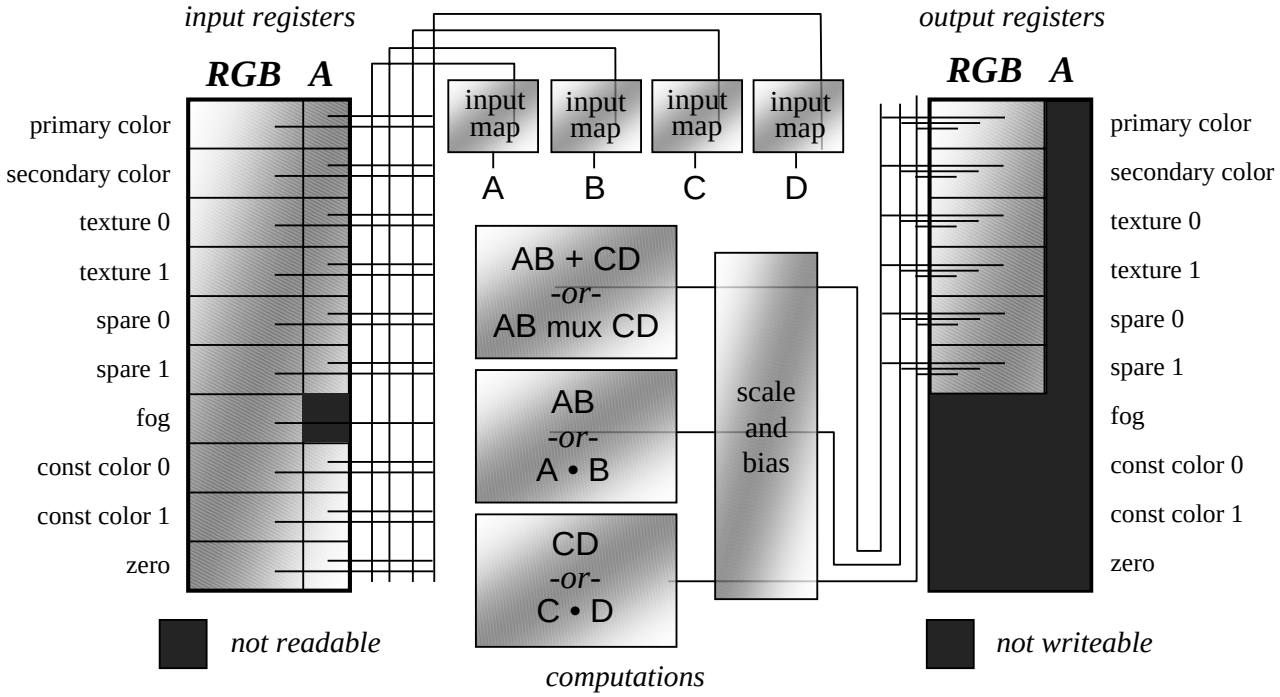
Programmierbare Rasterisierungseinheiten

- **Programmierbares Blending**
- **Programmierbare Texture-Lookups**
- **Indirektion**
  - Nichtlineare Farbabbildungen, z.B.
    - zwischen RGB und Lab, Luv
    - Farbtontransformationen (tone mapping)
  - Texturadvektion
- **Historie:**
  - Register Combiners (nVidia)
  - Texture Shaders (nVidia)
  - Heute: Pixel Shader / Fragment-Programme

Erweiterte OpenGL Pipeline

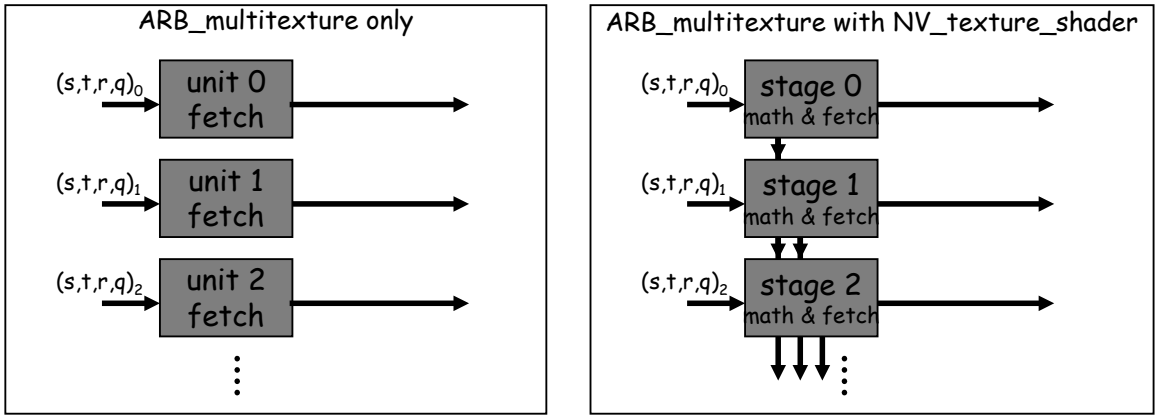


NVidia Register Combiner



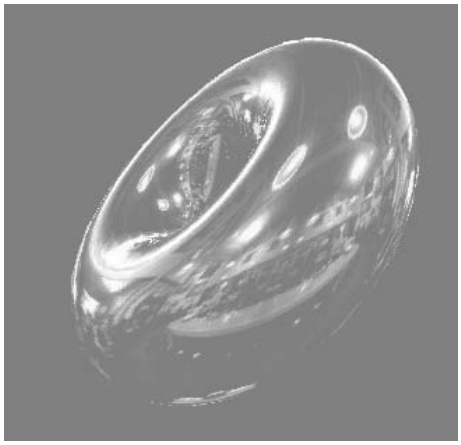
NVidia Texture Shader

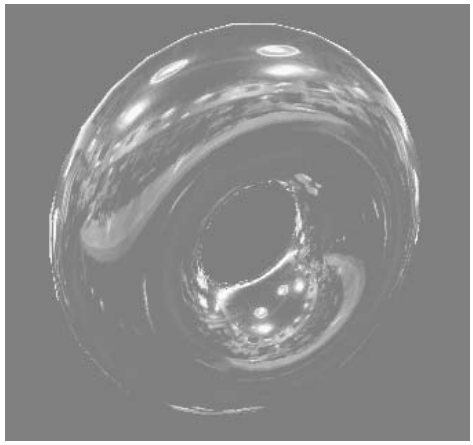
- **Texture Shader ersetzen Standard Fetch**
- **Ergebnisse früherer Stufen können in nächsten Stufen verwendet werden**
- **Ermöglicht indirekte Adressierung**



Reflection / Environment-Mapping

- **Darstellung reflektierender Objekte, z.B. Chrom**
- **Pseudo-Ray-Tracing**
  Verfolgung des reflektierten Strahls
- **Approximation der Reflexion ohne direktes Ray-Tracing**
- **Speichere Umgebung in Textur**





Reflection-Mapping

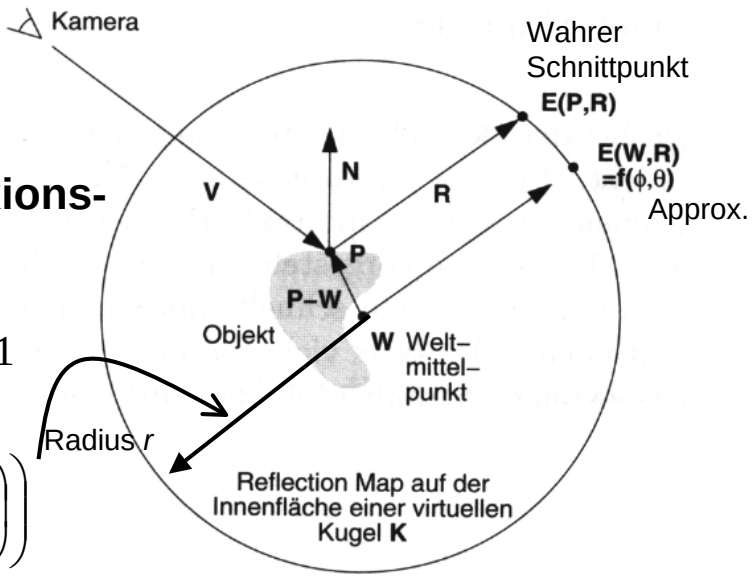
- **Virtuelle Kugel um Objekt: Reflection-Map**
- **Reflexionsvektor aus Sichtvektor und Normale**
  $R = V - 2\,(N \bullet V)N$
- **Annahme:**
  $r >> | P - W |$
- **Konvertiere Reflexions-richtung (φ,θ) in Texturkoordinaten**

$$R \text{ mit } R_x^2 + R_y^2 + R_z^2 = 1$$

$$\text{und } -1 < R_{x,y,z} < +1$$

$$s = \frac{1}{2}\left(1 + \frac{1}{\pi}\tan^{-1}\left(\frac{R_x}{R_y}\right)\right)$$

$$t = \frac{1}{2}\left(R_z + 1\right)$$

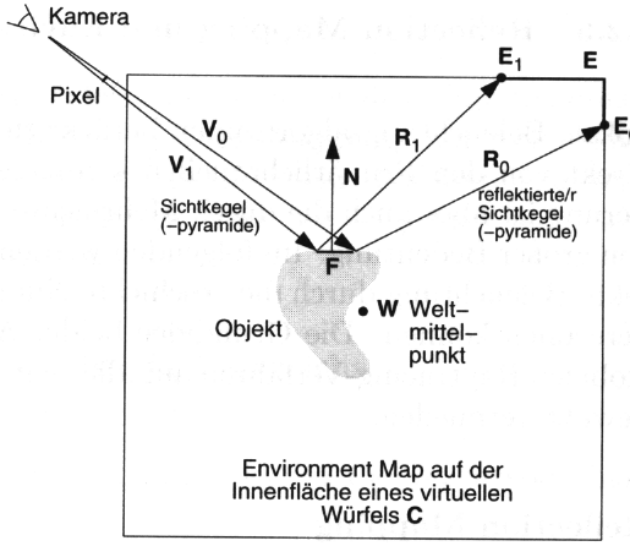


Reflection-Mapping: Grenzen

- **Umgebung (unendlich) weit weg**
- **Spiegelndes Objekt kompakt und begrenzt**
- **Keine Selbstreflexion des Objekts**
- **Zugriff auf nur einen Abtastpunkt, damit Aliasing**
- **Keine sphärische Interpolation (also pro Frag.)**
- **Aufwändige Umrechnung eines Bildes in sphärische Koordinaten**

Environment-Mapping

- **Direkte Erweiterung des Reflection-Mapping:**
  - Sichtpyramide (View Frustum) statt Einzelstrahl
  - Virtueller Würfel statt virtueller Kugel



Environment-Mapping *cont.*

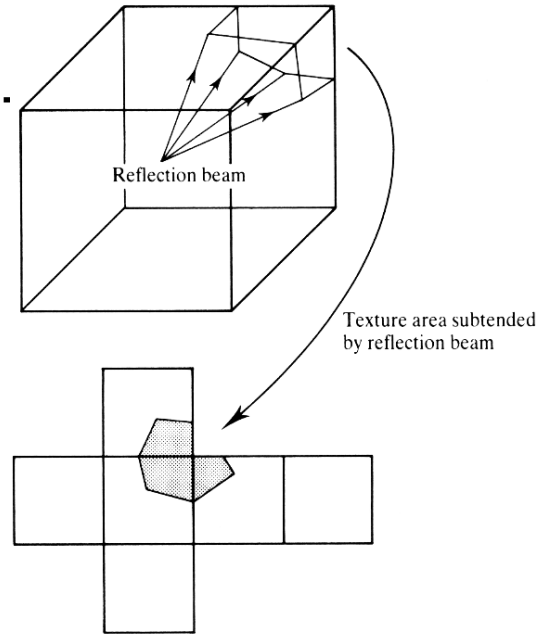
- **Vorteile:**
  - Anti-Aliasing, einfache Vorfilterung auf Würfelebenen
  - Umrechnung in sphärische Koordinaten entfällt
- **Variationen: Weitere Abbildungen möglich**
  - Cube map - Würfel
  - Sphere map - Kugel
  - Parabolic Map – Paraboloid
- **Nachteile/Grenzen:**
  - Zugriff nur über 3D Orientierung
  - Umgebung (unendlich) weit weg
  - Spiegelndes Objekt kompakt und begrenzt
  - Keine Selbstreflexion des Objekts
  - Lineare Interpolation des Reflexionsvektors

Environment-Mapping: Beispiel

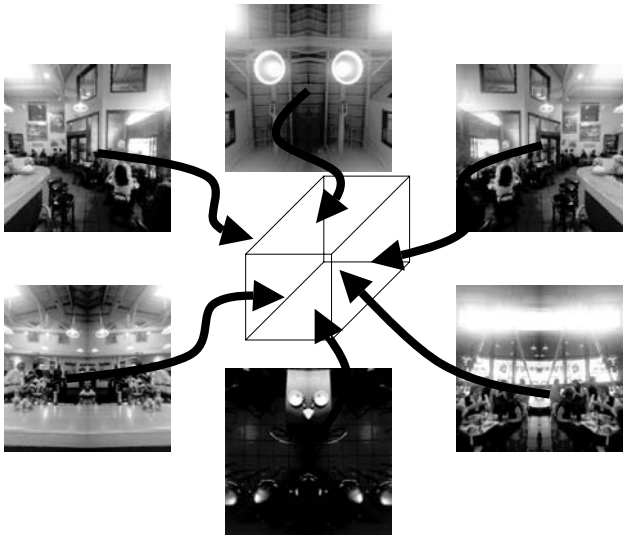
- **Terminator 2**

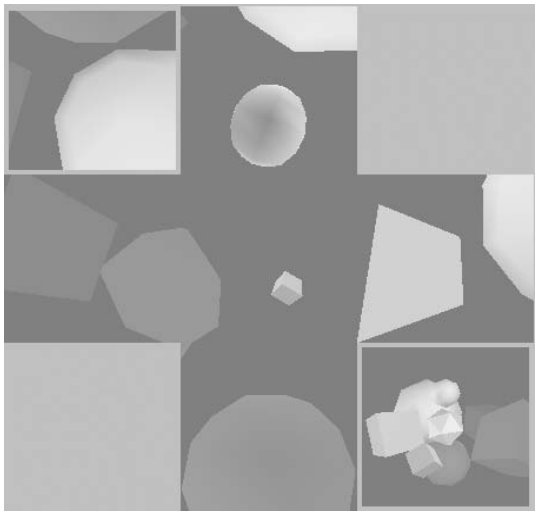
Cube-Mapping

- **Cube-Environment-Map, Box-Map**
  - Aufteilung in 6 Seitenflächen (Environment-Texturen)
  - blickpunkt*unab*hängig
  - Hardwareunterstützung
- **Einfache Texturkoordinaten-Generierung aus R**
  - betragsmäßig größte Komponente wählt Seite
  - übrige Koordinaten auf [0,1] skalieren
  - Beispiel: (-0.2,0.6,-0.8) wählt –z-Seite und dort (0.4,0.8)



Cube-Maps aus Bildern und Rendering





Cube-Map in OpenGL

- **ARB-Erweiterung oder ab OpenGL 1.3**

```
glEnable(GL_TEXTURE_CUBE_MAP_ARB)
glTexImage(GL_TEXTURE_CUBE_MAP_POSITIVE_X_ARB,
  level, format, ...)
```

ebenso die 5 anderen Seiten

```
glTexGen(GL_S,GL_TEXTURE_GEN_MODE,
  GL_REFLECTION_MAP_ARB)
```

ebenso für GL_T, GL_R

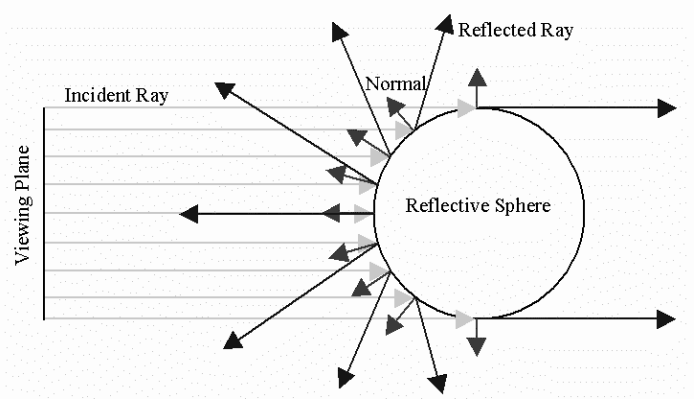
```
glEnable(GL_TEXTURE_GEN_S)
```

ebenso T und R

```
glEnable(GL_NORMALIZE)
```

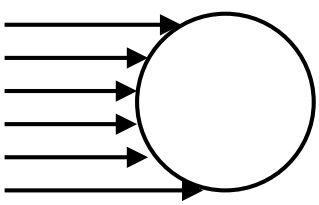
Sphere-Mapping

- **einfachster Ansatz mit nur einer 2D Textur**
- **Bild der ganzen Umgebung auf einer kleinen Chrom-Kugel wird zur Environment-Map**
- **blickpunkt*ab*hängig**
- **ungleiche Abtastung**
- **Rand ist Singularität**



Sphere-Mapping: Erzeugung

- **Erzeugung der Sphere-Maps (Original 1982/83)**
  - z.B. Aufnahme einer reflektierende Kugel



Sphere-Mapping in OpenGL

- **Texgen-Modus GL_SPHERE_MAP berechnet Reflexionsvektor R pro Vertex in Augkoordinaten aus transformierter Normale und Sichtvektor**
- **konvertiert diesen in Texturkoordinaten**

$$R = \begin{pmatrix} R_x \\ R_y \\ R_z \end{pmatrix} \text{ mit } R_x^2 + R_y^2 + R_z^2 = 1 \text{ und } -1 < R_{x,y,z} < +1$$

$$m = \sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

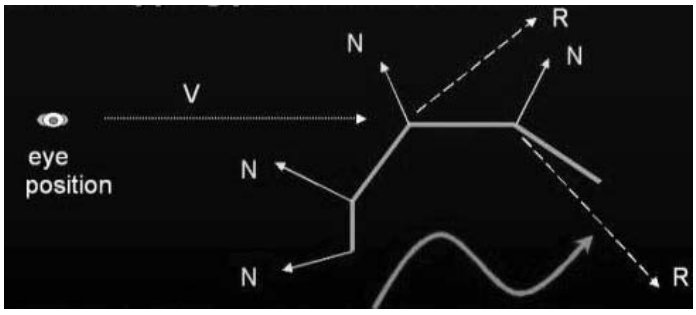
$$s = R_x / 2m + 0.5$$

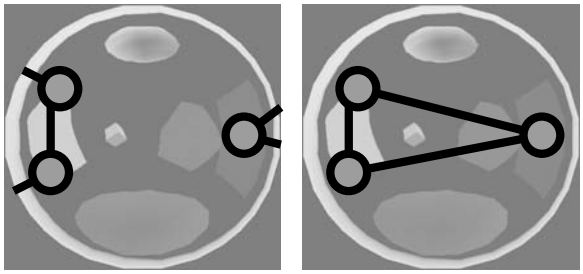
$$t = R_y / 2m + 0.5$$

```
glTexGen(GL_S,GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP)
```

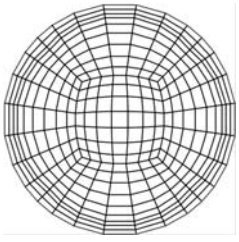

Sphere-Mapping: Probleme

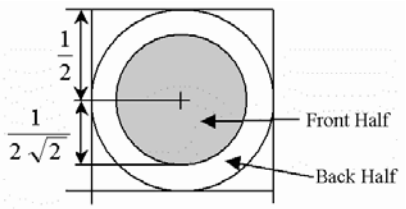
- **Artefakte am Rand (streifender Einfall)**
- **Verwende mehr Polygone bei hoher Krümmung**
- **Keine Interpolation über den Rand**





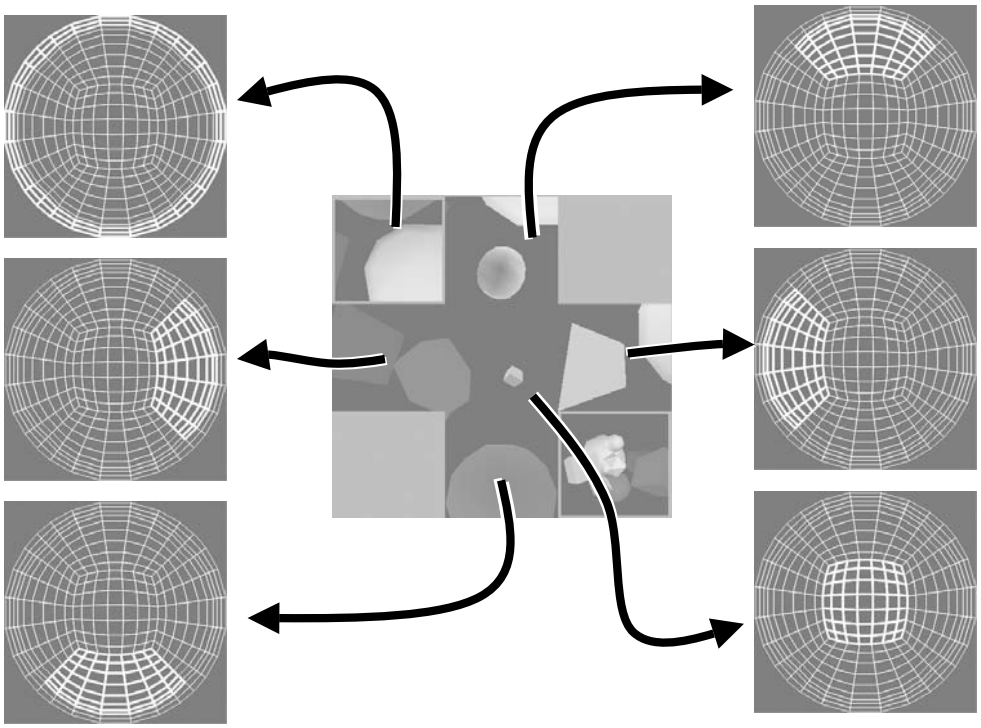
- **Eigentlich Dreieck mit gekrümmten Kanten (sphärisch)**
- **Verzerrung deutlich im Würfelgitter**





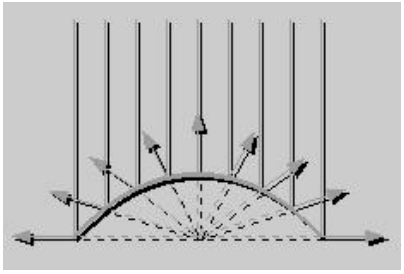
Sphere-Map aus Cube-Map

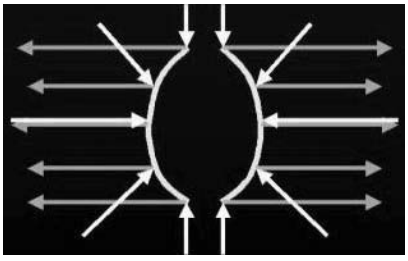
- **Verwende inverse Abbildung**

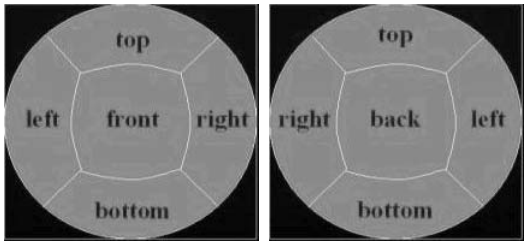


Parabolische Abbildung

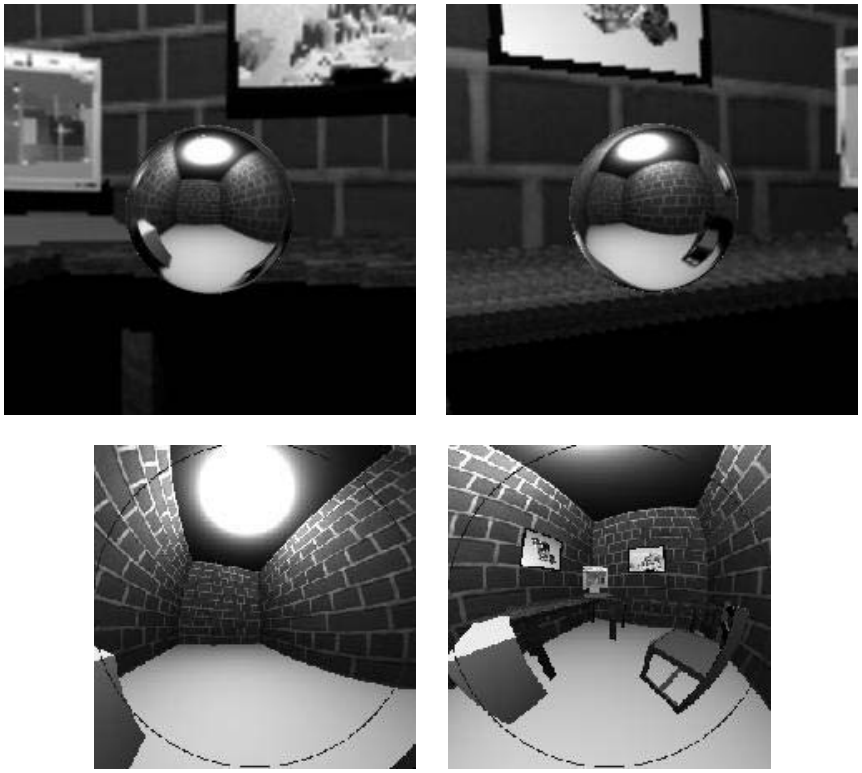
- **Aufteilung in 2 Bilder (Vorder-/Rückseite)**
- **wenig Verzerrung am Rand**
- **blickpunkt*unab*hängig**
- **2-Pass bzw. 2-Textur-Verfahren**
- **Generierung des Eye-Space Reflection Vector als Textur-koordinate pro Vertex**
- **Dann Transformation durch Texturmatrix**





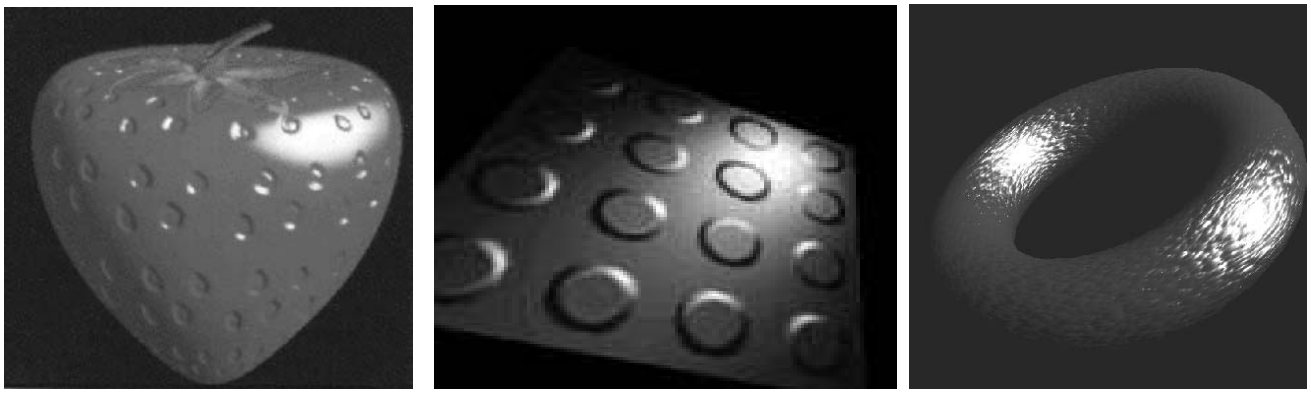


Parabolische Environment-Maps



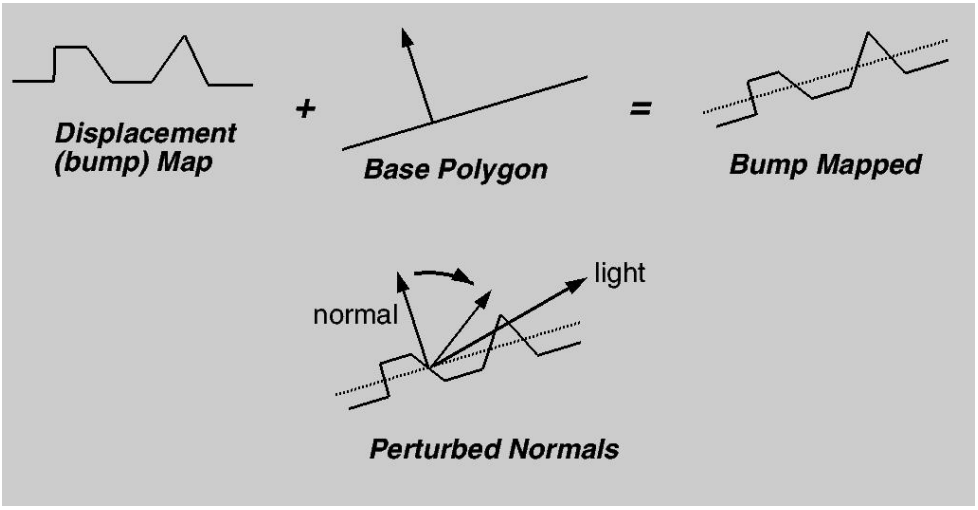
Bump-Mapping

- **Realistische Oberflächen sind nicht perfekt glatt**
- **Rauhigkeit durch Bildtextur nicht möglich**
- **Normalen-Variation führt zu Tiefeneindruck**
- **Keine Änderung der Geometrie! (Silhoutte!)**



Bump-Mapping

- **Modulation des Normalenvektor berechnet aus Veränderung der Basisfläche durch Bump-Map**
- **Fläche bleibt geometrisch flach**
- **Normalen variieren pro Pixel**



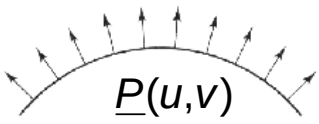
Bump-Mapping: Hintergrun

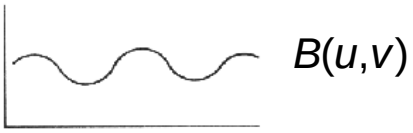
- **Fläche (z.B. Bezier-Patch)** $\underline{P}(u,v)$
- **mit Normale** $\underline{N}(u,v) = \frac{\partial \underline{P}}{\partial u} \times \frac{\partial \underline{P}}{\partial v}$
- **mit Bump-Map gestörte Fläche**

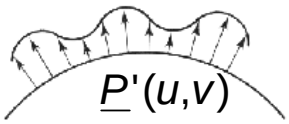
  $\underline{P}'(u,v) = \underline{P}(u,v) + B(u,v) \cdot \underline{N}(u,v)$
- **wie ändert sich die gestörte Normale?**

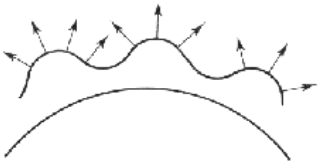
  $\underline{N}'(u,v) = \frac{\partial \underline{P}'}{\partial u} \times \frac{\partial \underline{P}'}{\partial v} =$

  $\underline{N} + \frac{\partial B}{\partial u}\left(\underline{N} \times \frac{\partial \underline{P}}{\partial v}\right) - \frac{\partial B}{\partial v}\left(\underline{N} \times \frac{\partial \underline{P}}{\partial u}\right)$
- **Braucht diskrete Ableitung der Bump-Map und lokale Tangentialebene der Fläche**



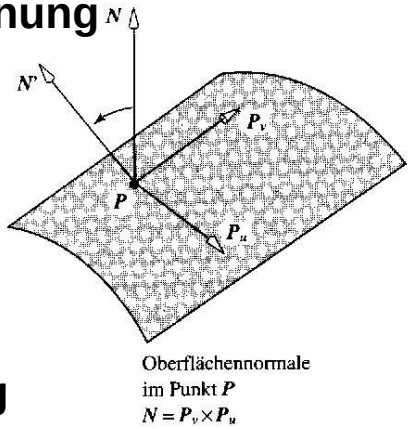




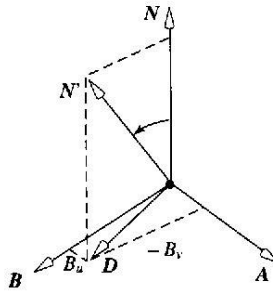


Bump-Mapping: Tangent Space

- **Änderung der Normale im „Tangentenraum“**
  $\underline{N}' = \underline{N} + \underline{D} \quad \underline{D} = B_u(\underline{N} \times \underline{P}_v) - B_v(\underline{N} \times \underline{P}_u) = B_u \underline{A} - B_v \underline{B}$
- **Beleuchtungberechnung pro Fragment (Phong Shading)**
- **Tangent-Space**
  - Vertex (0,0,0)
  - Normale (0,0,1)
  - Tangente
- **Diffuse Beleuchtung**
  $\underline{N}' \bullet \underline{L} = \underline{N} \bullet \underline{L} + \underline{D} \bullet \underline{L}$
- **Transformiere Lichtquellenrichtung in Tangent-Space und berechne Skalarprodukt (Winkel) dort**



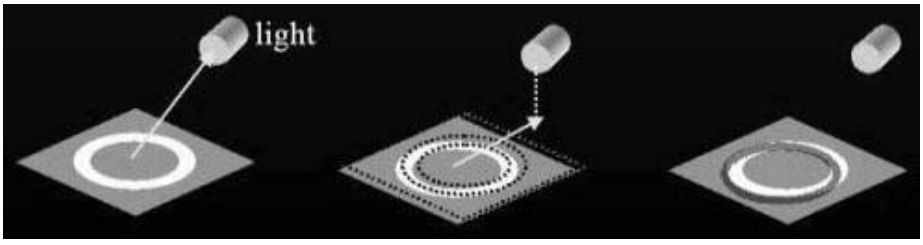
Oberflächennormale im Punkt P
$N = P_v \times P_u$



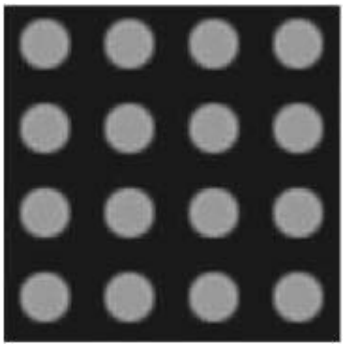
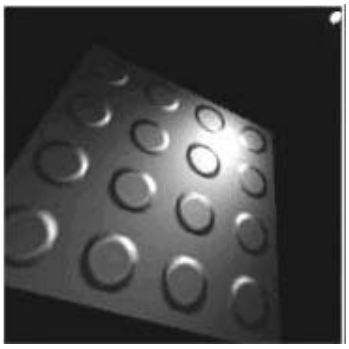
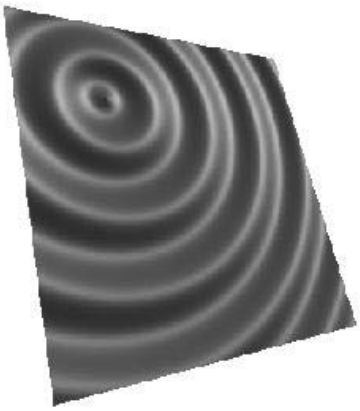
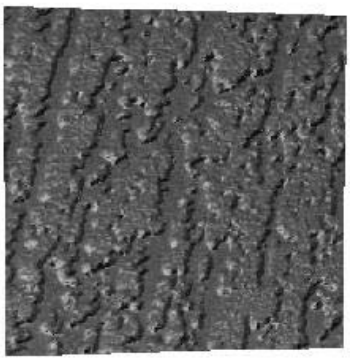
D ist gegeben durch:
$D = B_u A - B_v B$

Normal-Maps

- **Vorberechnung der modifizierten Normalen N‘**
- **Abspeichern in Textur (RGB)=(N_x,N_y,N_z)**
- **Berechnung der diffusen Beleuchtung <$\underline{N} \bullet \underline{L}$> pro Pixel in der Fragment-Verarbeitung**
- **z.B. mit Farbmatrix oder Fragment-Programm**
- **Per-Vertex-Vektor zur Lichtquelle wird interpoliert**
- **Embossing bzw. Tangent-Space-Shading**
  - $\underline{D} \bullet \underline{L}$ ist die Änderung von Bump-Map B in Richtung von L
  - Verschiebe Geometrie in Richtung der Lichtquelle und subtrahiere



Ergebnisse

Normal Maps – Relief Maps


Color


Height


Normal

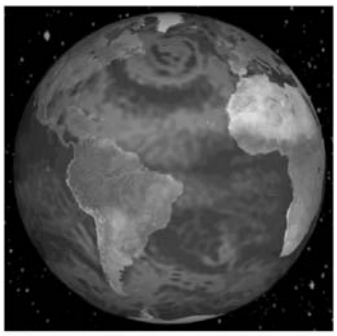

Normal - x

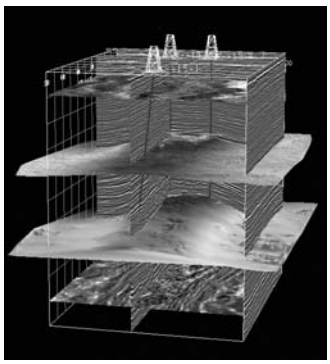

Normal - y

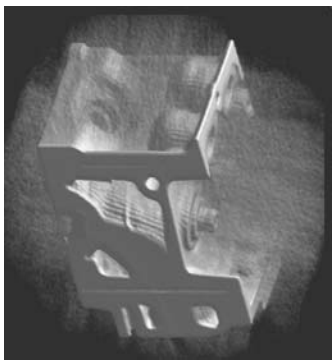

Normal - z

3D-Texturen und Volumenvisualisierung

- Ausgangsdaten sind 3D
  - Röntgenabsorption im Körper/Material
  - Feuchtigkeit in Atmosphäre
  - Dichteverteilung im Erdinneren
- Daten oft auf uniformen 3D-Gitter Millionen von Zellen (Voxel)
- Problem: Verdeckung

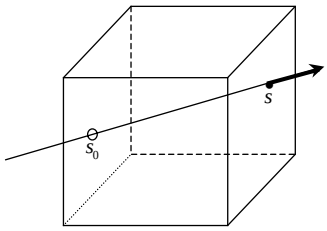






Direkte Volumenvisualisierung

- **Idee: innere Struktur sichtbar durch Transparenz**
- **Physikalisches Modell: Sternennebel Volumen als selbstleuchtendes absorbierendes Gas**
- **Strahlungstransportgleichung ohne Streuung**
- **Strahlverfolgung:**
  - pro Bildpunkt ein Strahl durch das Volumen
  - interpoliere Daten trilinear an Stützstellen
  - akkumuliere Intensität gewichtet mit Transparenz
- **Transferfunktion: Zuordnung von Farbe und Transparenz zu Datenwerten**
- **Unterschiedliche Visualisierungsstile**

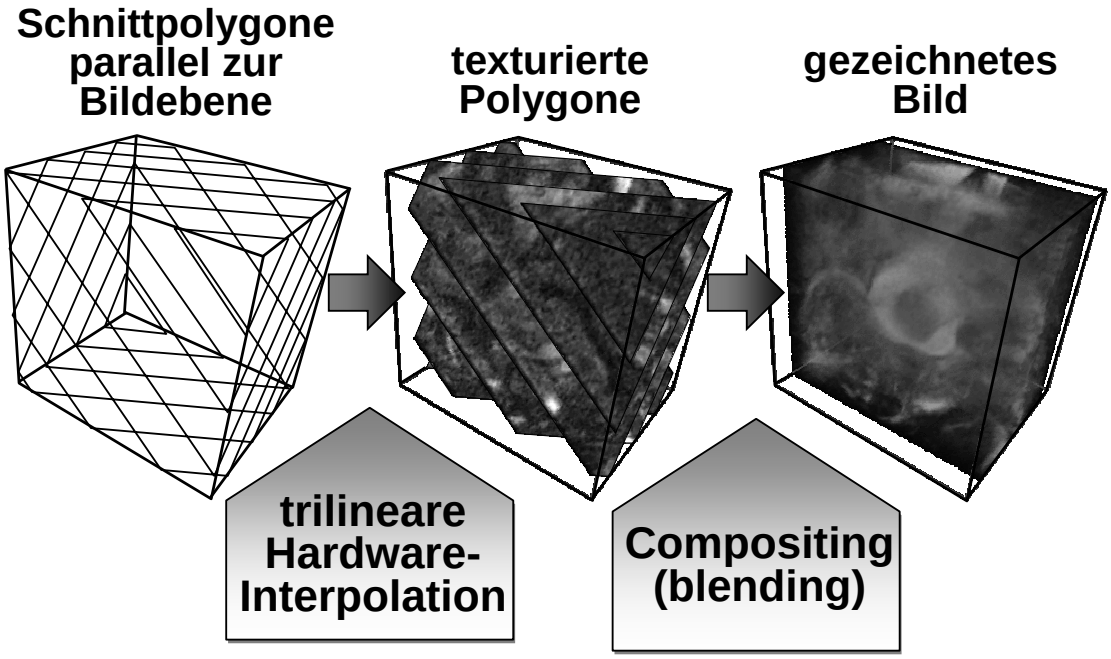


Verschiedene Transferfunktionen

Verschiedene Transferfunktionen *cont.*

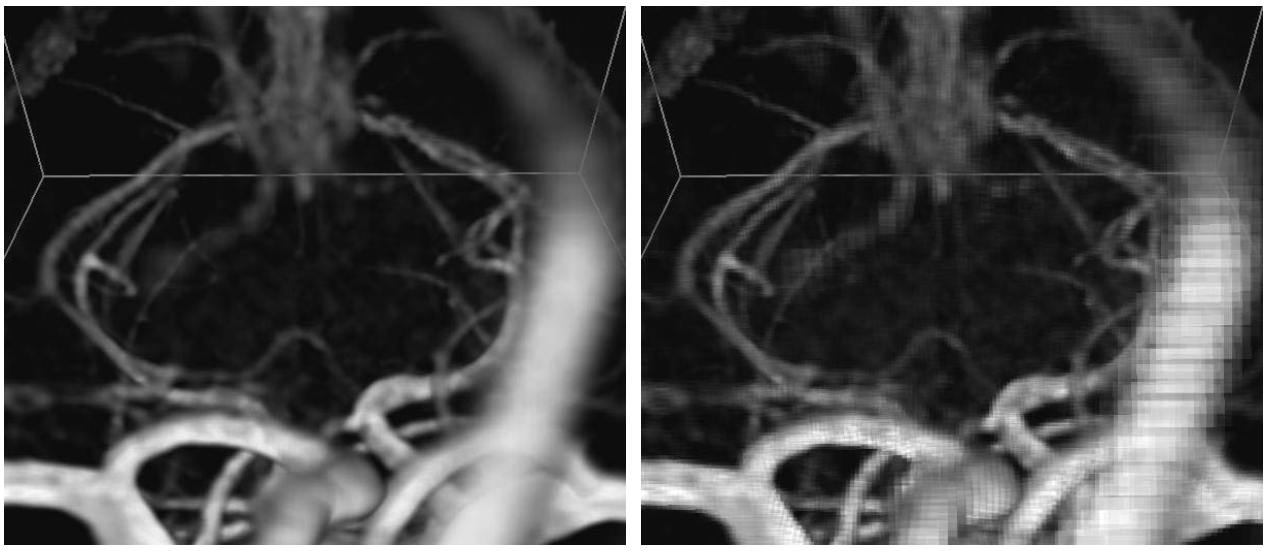
Texturbasierte Volumenvisualisierung

- **Speichere Daten in 3D-Textur**



Interpolation bei Volumenvisualisierung

- **Nearest-Neighbor (0. Ordnung)**
- **Trilinear (1. Ordnung)**

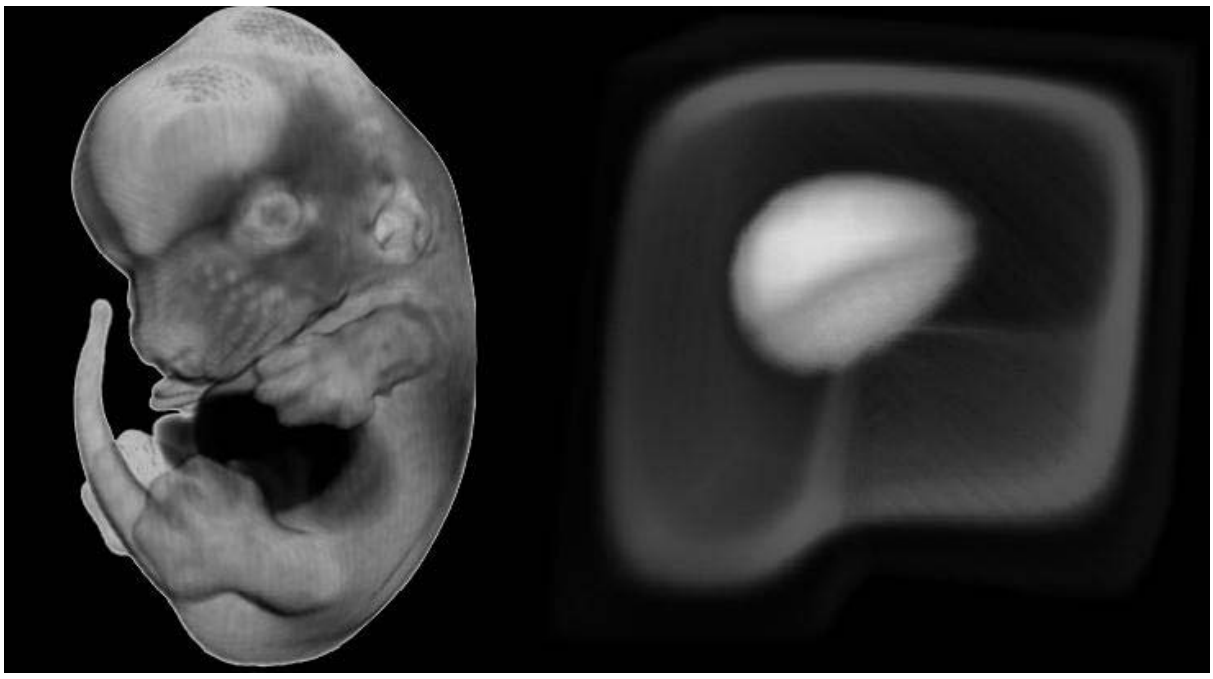


Trilinear **Nearest-Neighbor**

Klassifikation und Volumenbeleuchtung

- **Pre-Shading (Präklassifikation)**
  - Erzeuge RGBA Volumentextur aus Skalarfeld
  - Farbindizes: Texturpalette
- **Post-Shading (Postklassifikation)**
  - Speichere Daten in Luminanz-Textur
  - Benötigt Lookup in Transferfunktion nach Interpolation
  - Spezielle Color-Table-Extension (SGI)
  - Dependent Texture in Fragmentprogramm
- **Volumenbeleuchtung**
  - Diffuse und spekulare Beleuchtung jedes Voxels
  - Benötigt 3D Normalentextur

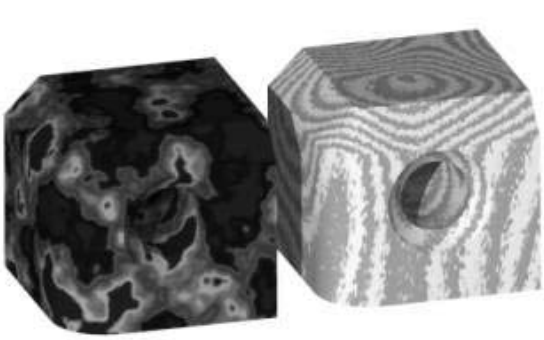
Volumenvisualisierung: Beispiele

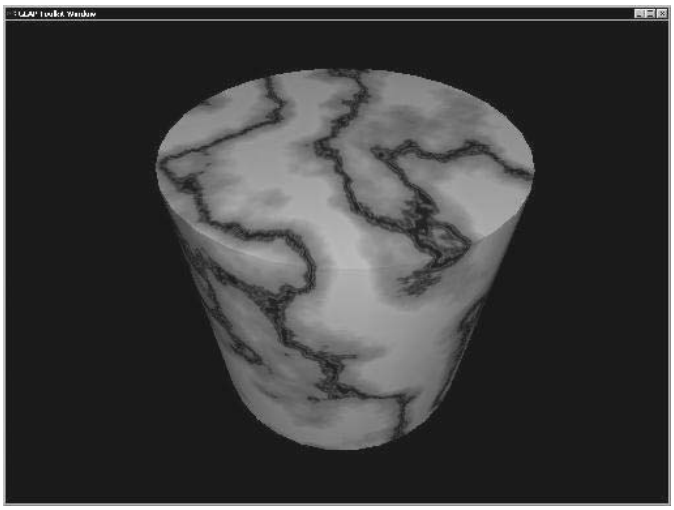


3D-Texturen für Oberflächen

- **Probleme von 2D-Texturen**
  - Tapeten-/Furnier-Effekt („oberflächlich“)
  - starke Texturverzerrung bei großer Flächenkrümmung
  - Texturkoordinaten schwierig für komplexe Topologie
- **Solid-Textures (3D-Texturen)**
  - z.B. Holzblock, Marmorblock
  - Herausschneiden einer Skulptur durch Zuweisung 3D Texturkoordinaten an Vertizes
- **Nachteile**
  - Gewinnung der 3D-Textur aus Schichten
  - großer Speicherplatzbedarf

Beispiel Solid-Textures





Bedeutung von Texture-Mapping

„Texture Mapping as a Fundamental Drawing Primitive“ (Haeberli, Segal 1993):

- **Image Warping**
- **Transparency Mapping, Surface Trimming**
- **Farbkonvertierung**
- **Phong-Shading**
- **Volumenvisualisierung**
- **....**