

Texturen und Shader

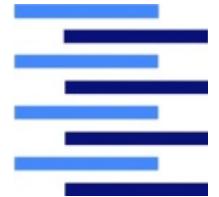


```
/*
 * Simple class to show summing functionality. TUTORIAL
 * Author: Philipp Jenke
 */
class Sum {
    /**
     * With this method you can compute the sum of the natural numbers from B to
     * a given number (A).
     */
    public static void main(String[] args) {
        int a; // A
        int b; // B
        int ergoenis; // C.
        int zaehler; // D.
        // Initialization
        n = 4; // 1.
        ergoenis = 8; // 2.
        zaehler = 1; // 3.
        // Computation
        while (zaehler <= n) { // 4.
            ergoenis = ergoenis + zaehler; // 4a)
            zaehler = zaehler + 1; // 4b)
        }
        // Print result
        System.out.println(ergoenis); // 5.
    }
}
```

Einführung in die Computergrafik

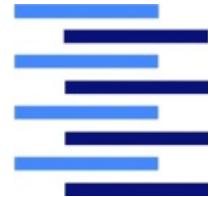
WS 2013/14, Prof. Dr. Philipp Jenke





Agenda

- > Texturen
 - > Einführung
 - > Interpolation
 - > Aliasing
 - > Erzeugen von Texturkoordinaten
 - > Weitere Anwendungen für Texturen
- > Shader



Einführung

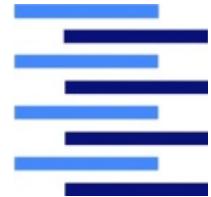
- > Realität: großes Spektrum geometrischer Formen und physikalischer Materialien
 - > Maserungen von Holz, Marmor, Tapeten
 - > Wolken
 - > Strukturen in unebenen Oberflächen (z.B. Rauputz, Apfelsinen, Baumstämme)
 - > im Hintergrund: Häuser, Maschinen, Pflanzen, Personen
- > exakte Nachbildung der geometrischen Formen?
 - > viel zu aufwändig



Einführung

- > Idee: Texturierung
 - > komplexe Gestaltung des Erscheinungsbildes von Objekten
 - > Modellierung der Wand als planare Oberfläche (genauso: Spiegel, Fenster, ...)
 - > Tapezieren der Fläche durch ein Bild





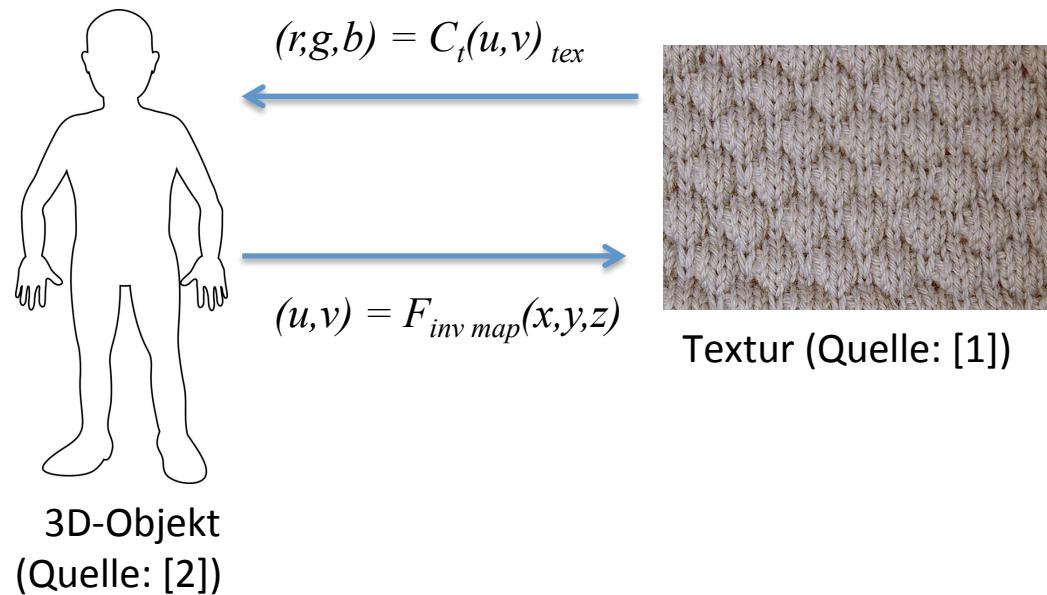
Einführung

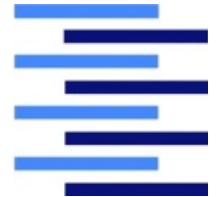
- > 2D-Texturen sind Funktionen, die Punkte der (u,v) -Texturebene auf (r,g,b) -Farben abbilden
 - > $(r,g,b) = C_{text}(u,v)$
- > Abbildung (Mapping) beschreibt, wie eine 2D-Textur auf einer Fläche aufgebracht wird
- > Visualisierung: Inverse-Mapping Problem lösen
 - > zu bekannten (x,y,z) -Koordinaten müssen passende (u,v) -Koordinaten gefunden werden
 - > $(u,v) = F_{inv\ map}(x,y,z)$



Einführung

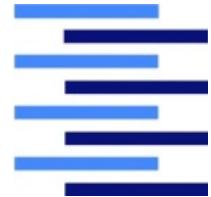
- > Abbildung zwischen Objektraum und Texturraum





Einführung

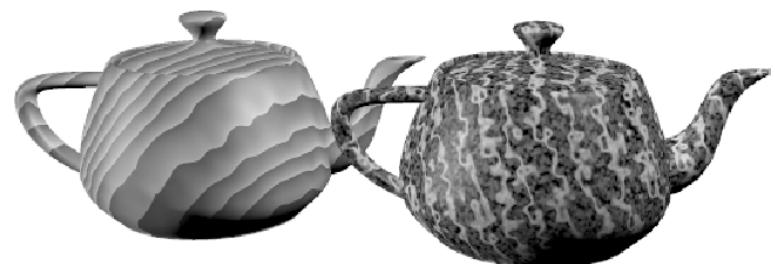
- > Texturierung einer Fläche mit einer 2D-Textur
 - > mathematisch: Hintereinander-Ausführung der beiden Abbildungen
 - > $(r,g,b) = C_{tex}(F_{inv\ map}(x,y,z))$



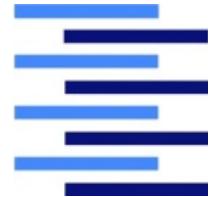
Einführung

Ausflug: 3D Texturen

- > Funktionen zur Abbildung von 3D-Punkten
 - > $(r,g,b) = C_{tex}(u,v,w)$
- > werden auch als Festkörpertexturen bezeichnet
- > Beispiele
 - > Holz, Marmor
- > Inverses Mapping
 - > Abbildung: $(r,g,b) = C_{tex}(u,v,w)$
- > Interpretation
 - > Herausschneiden eines Körpers aus dem Texturkörper
- > Vergleiche
 - > Dichtefunktion (implizite Funktionen)

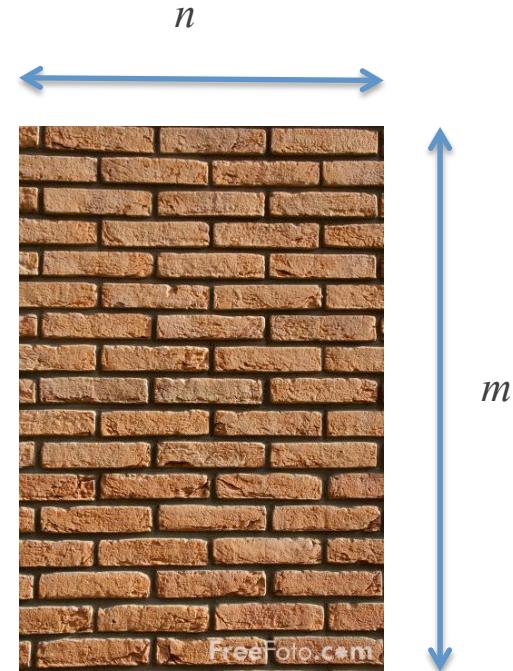


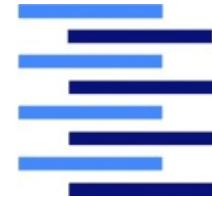
Einführung



Aufbau von Texturen

- > Raster von Farbwerten
- > 2D: Bild mit Breite n und Höhe m
 - > $\{C[i,j] | 0 \leq i \leq n, 0 \leq j \leq m\}$
- > jeder Eintrag
 - > Vektor mit drei Komponenten (r,g,b)
 - > wird als Texel bezeichnet





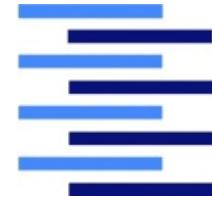
Einführung

Erzeugen von Texturen

- > Vorrat nahezu unerschöpflich
 - > Fotoapparate
 - > Scanner
 - > Speichermedien
- > Generieren von komplexen 2D-Texturen
 - > einfach
 - > photorealistische Ergebnisse

Photorealistische Texturen (Quelle: [3])





Einführung

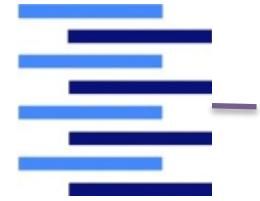
Herausforderungen

- > große Bilder haben hohen Speicherbedarf
- > beim Vergrößern kleiner Bilder treten Artefakte auf
- > Wiederholung (Fortführung von Texturen oft kompliziert)

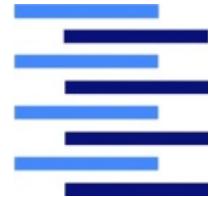


Wiederholbare
Textur
(Quelle: [4])

- > Kontext der Bilder passt oft in 3D-Anwendung nicht mehr
 - > Sonnenstand
 - > Schattenwurf
 - > ...

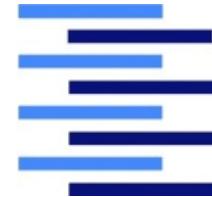


Interpolation



Interpolation und Aliasing

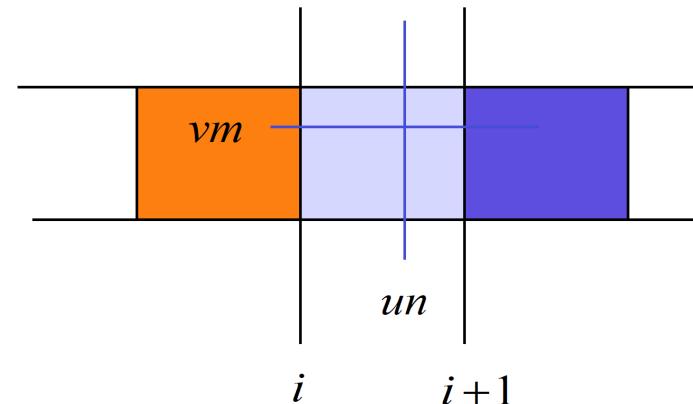
- > Verzerrungen beim Mapping
- > Notwendigkeit, Texturwerte an beliebigen Positionen zu rekonstruieren
 - > Bereichsüberschreitungen
 - > Interpolation
 - > Filterung

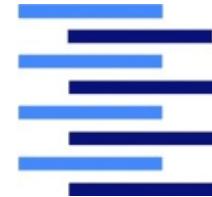


Interpolation

- > Interpolation – Nächstgelegener Nachbar

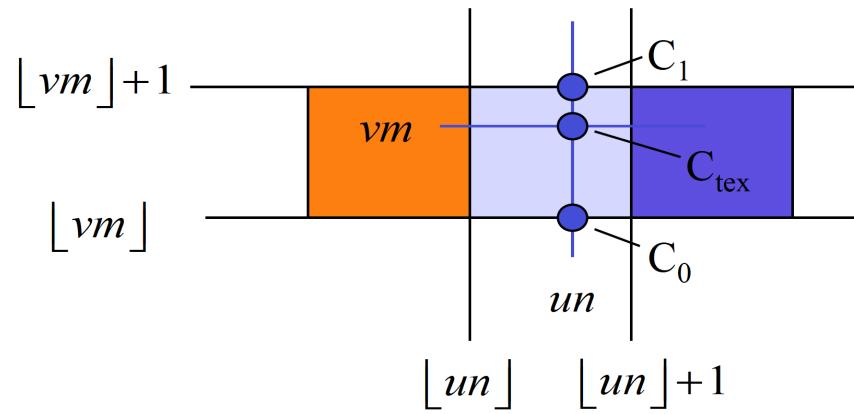
$$C_{tex}(u, v) = \begin{cases} C[\lfloor un \rfloor, \lfloor vm \rfloor] & : u < 1, v < 1 \\ C[\lfloor n-1 \rfloor, \lfloor vm \rfloor] & : u = 1, v < 1 \\ C[\lfloor un \rfloor, \lfloor m-1 \rfloor] & : u < 1, v = 1 \\ C[\lfloor n-1 \rfloor, \lfloor m-1 \rfloor] & : u = 1, v = 1 \end{cases}$$





Interpolation

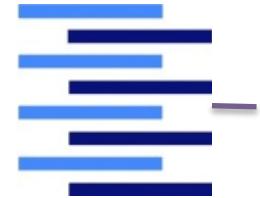
> Interpolation – Bilineare Interpolation



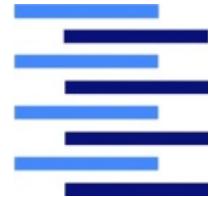
$$C_0(u, v) = \hat{u} * C[\lfloor un \rfloor + 1, \lfloor vm \rfloor] + (1 - \hat{u}) * C[\lfloor un \rfloor, \lfloor vm \rfloor]$$

$$C_1(u, v) = \hat{u} * C[\lfloor un \rfloor + 1, \lfloor vm \rfloor + 1] + (1 - \hat{u}) * C[\lfloor un \rfloor, \lfloor vm \rfloor + 1]$$

$$C_{tex}(u, v) = \hat{v} * C_1(u, v) + (1 - \hat{v}) * C_0(u, v)$$

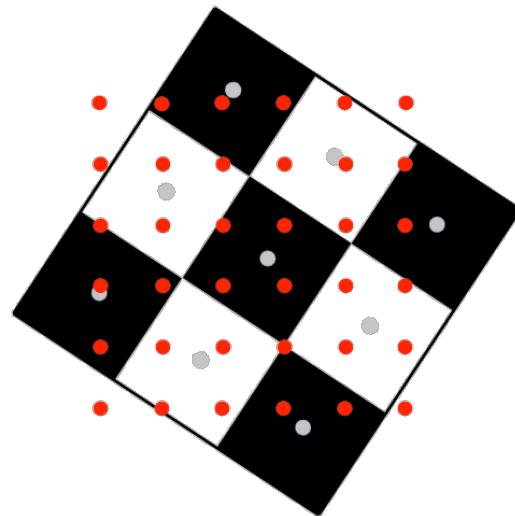


Aliasing

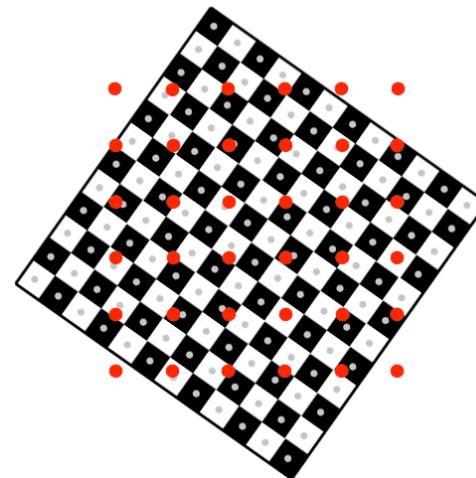


Aliasing

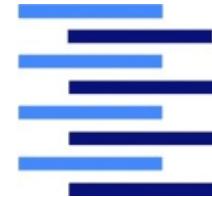
- > Herausforderungen: Zu viel/zu wenig Information in Textur für Bildschirmschardarstellung
 - > Textur: Schachbrettmuster im Hintergrund
 - > Pixel auf dem Bildschirm: rote Punkte



Vergrößerung
(zu wenig Texturinformation)

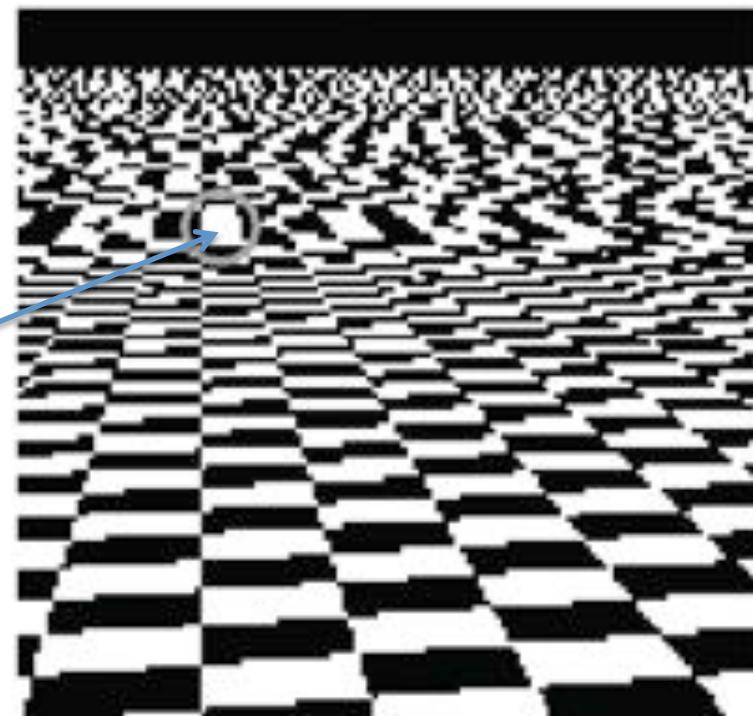


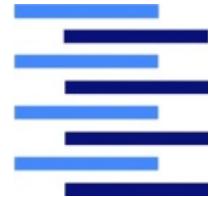
Verkleinerung
(zu viel Texturinformation)



Aliasing

- > Aliasing
- > Abbildung auf Bildschirmpixel für ästhetische Darstellung nicht ausreichen
- > Entstehung: Rundungsartefakte

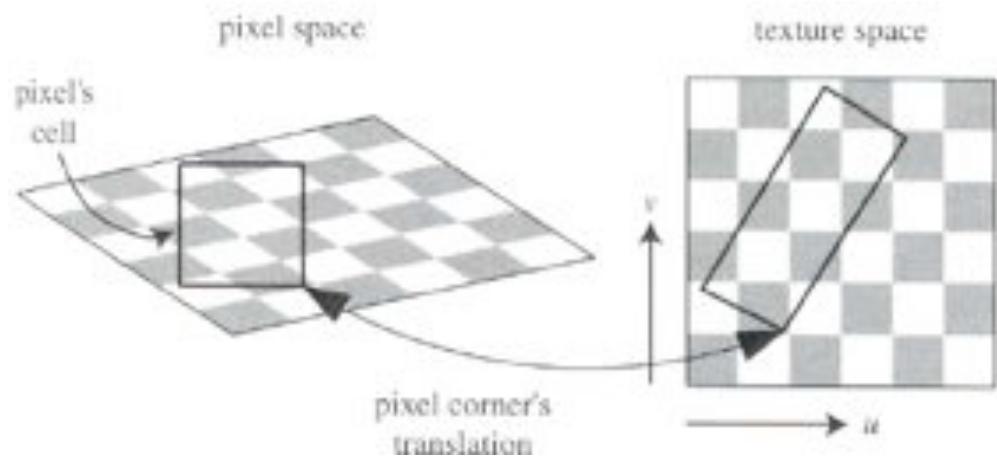
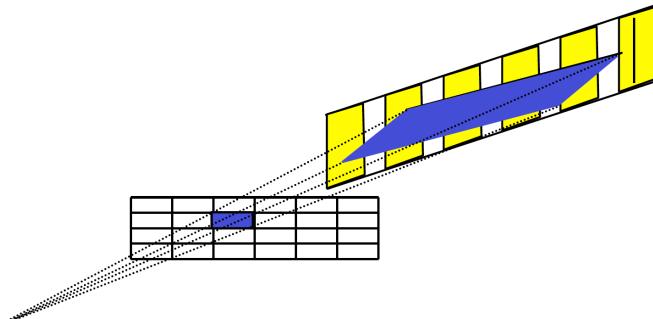


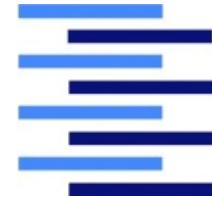


Aliasing

Lösungsidee

- > Berechnung des Footprints
 - > Abbildung des Parallelogramms, das der Pixel auf der Textur bedeckt
 - > Einsammeln der Texturwerte innerhalb des Footprints
- > Echtzeitanwendungen
 - > Vorberechnung durch Reduktion auf einfache Flächen
 - > Verwenden von Mip-Maps

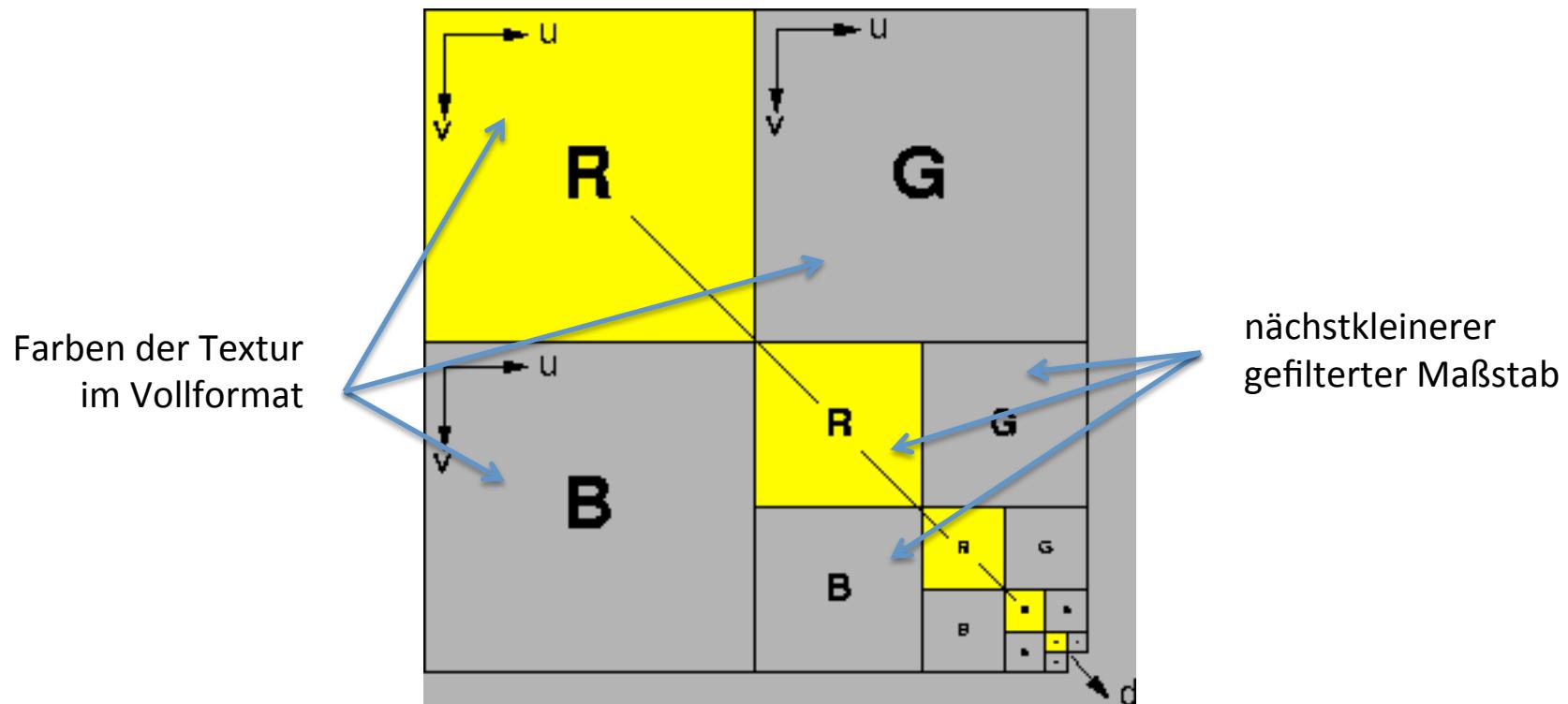


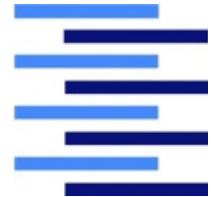


Aliasing

Mip-Map

- > speichert quadratische Textur der Größe $n \times n$, $n = 2^k$
- > jeweils in halbiertem (gefilterter) Auflösungsstufe

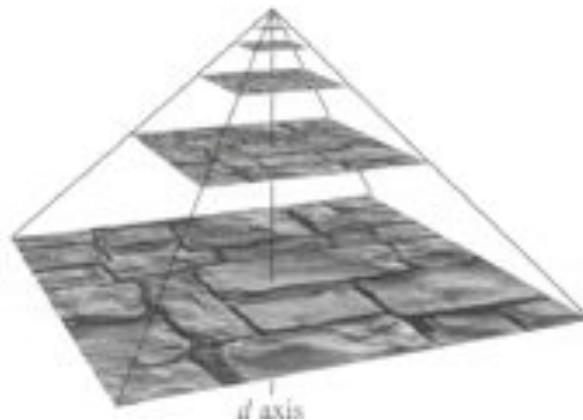


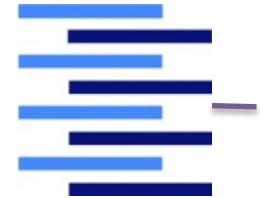


Aliasing

Mip-Map

- > Stufe 0
 - > Übernehmen der Texturwerte
 - > $C_{mip}^0[i,j] = C[i,j]$
- > übrige Stufen
 - > Filterung der vorangegangenen Stufe
 - > $C_{mip}^d[i,j] = 0.25 * (C_{mip}^{d-1}[2i,2j] + C_{mip}^{d-1}[2i+1,2j] + C_{mip}^{d-1}[2i,2j+1] + C_{mip}^{d-1}[2i+1,2j+1])$



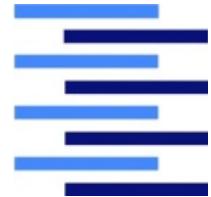


Erzeugen von Texturkoordinaten



Erzeugen von Texturkoordinaten

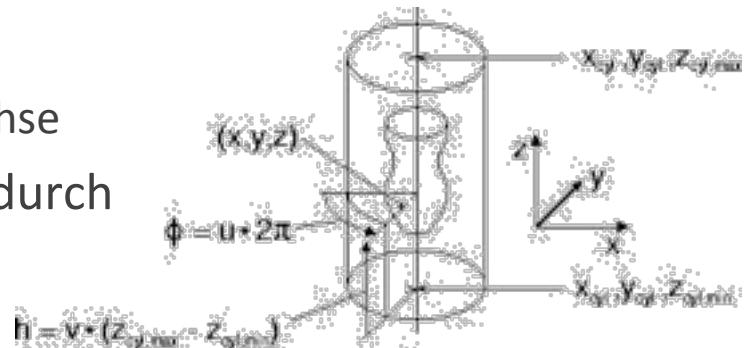
- > Zuordnung von Dreieckspunkten zu Texturkoordinaten bei einem gegebenen Modell
 - > Umgeben des Objektes mit einer einfachen parametrisierbaren Fläche
 - > Abbilden der 2D-Textur auf die umhüllende Fläche
 - > von dort auf die Objektoberfläche
 - > geeignete Flächen
 - > Zylinder, Kugeln, Quader
 - > direkte Abbildung bei synthetischen Texturen möglich

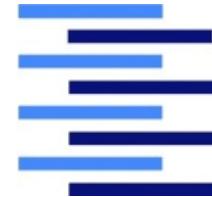


Erzeugen von Texturkoordinaten

Zylinder-Mapping

- > Umgeben des Objektes mit einem endlichen Zylinder
- > Vereinfachung der Berechnung
 - > Hauptachse parallel zur Zylinderhauptachse
- > Parametrisieren der Zylinderoberfläche durch
 - > Winkel φ
 - > Höhe h
- > Punkte (x, y, z) im Inneren werden senkrecht von der Zylinderachse auf Zylinderfläche projiziert
- > (φ, h) wird als (u, v) interpretiert

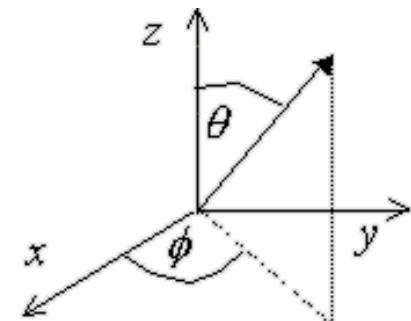




Erzeugen von Texturkoordinaten

Kugel-Mapping

- > Kugeloberfläche kann mit Kugelkoordinaten φ und θ parametrisiert werden
- > Mittelpunkte der Kugel (x_s, y_s, z_s)
- > Projektion vom Kugelmittelpunkt auf Kugeloberfläche
- > (φ, θ) wird als (u, v) interpretiert



$$u = \frac{\pi + \arctan 2 (y - y_s, x - x_s)}{2\pi}$$
$$v = \frac{\arctan 2 \left(\sqrt{(x - x_s)^2 + (y - y_s)^2}, z - z_s \right)}{\pi}$$





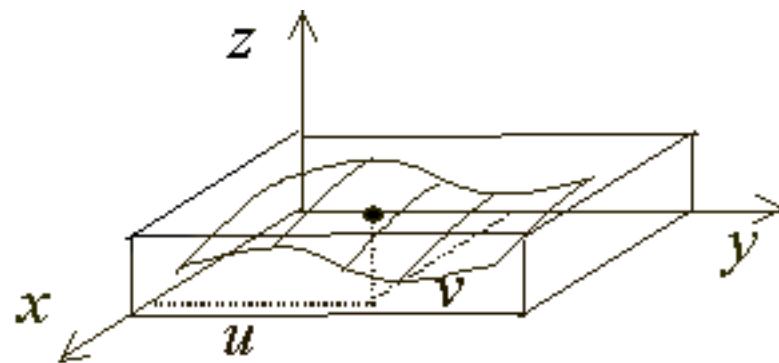
Erzeugen von Texturkoordinaten

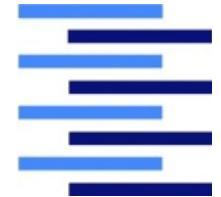
Box-Mapping

- > in der Regel: achsenparallele Bounding-Box des Objektes
 - > Längste Kante: u
 - > zweitlängste Kante: v

$$u = \frac{x - x_{box,min}}{x_{box,max} - x_{box,min}}$$

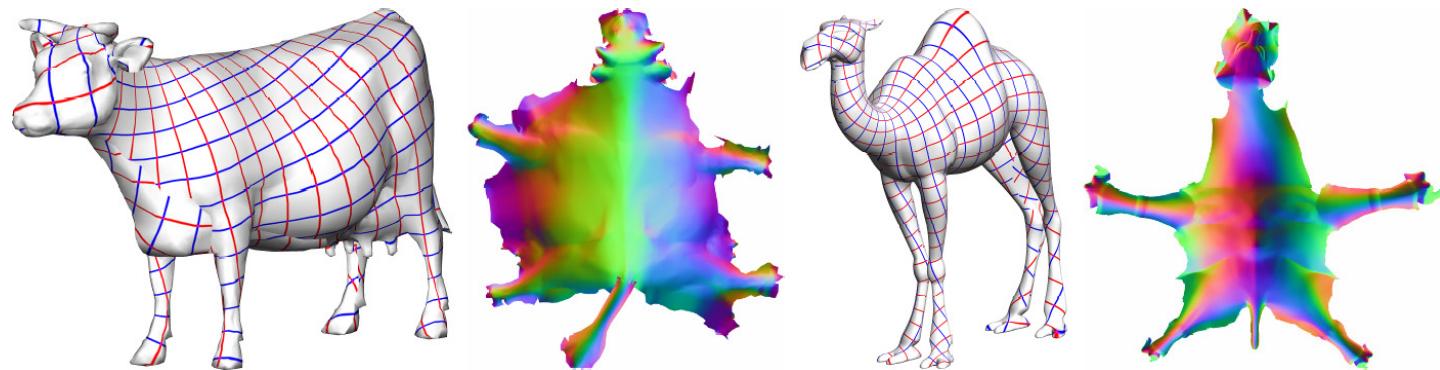
$$v = \frac{y - y_{box,min}}{y_{box,max} - y_{box,min}}$$



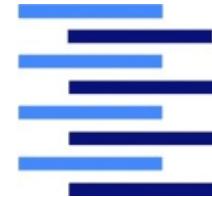


Erzeugen von Texturkoordinaten

- > Aktives Forschungsgebiet: Entwicklung von Algorithmen zur automatischen Parametrisierung von Oberflächen



Parametrisierung von Oberflächen (Quelle [6])

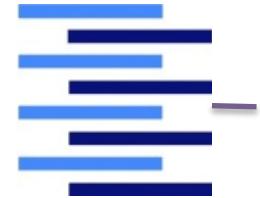


Textur-Atlas

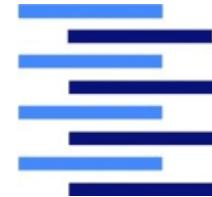
- > Sammeln mehrere Einzeltexturen in einem Texturbild
 - > sogenannter Texturatlas



Texturatlas
(Quelle [5])

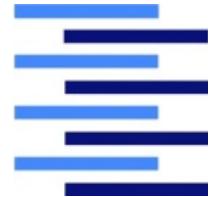


Weitere Anwendungen für Texturen



Weitere Anwendungen für Texturen

- > Texturen können weitere Oberflächeneigenschaften darstellen
 - > Light-Mapping
 - > Bump-Mapping
 - > Environment-Mapping

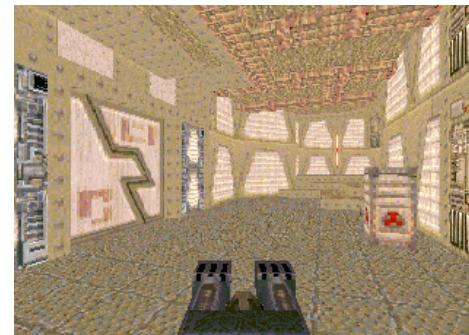


Light-Mapping

- > Beleuchtungsmodell vorberechnen
 - > z.B. Phong-Modell
- > Die Leuchtdichte L_{phong} ist wellenlängenabhängig
 - > muss zur Ausgabe in darstellbare C_{phong} umgerechnet werden
 - > Modulierung der Texturfarbe mit Lightmaps



+



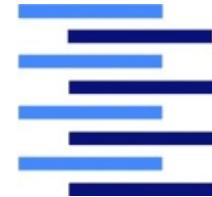
=



Light-Map

Texturen

Kombination
(Quelle: [7])



Environment-Mapping

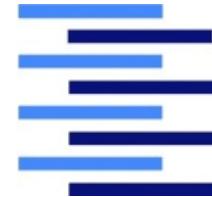
- > Approximation von Reflexionen über Texturen (hardwareunterstützt)
- > Idee
 - > reflektiertes Objekt ist klein im Vergleich zum Objekt
 - > einfallende Beleuchtungsstärke hängt nur von der Richtung ab
 - > nicht von der Position des Punktes auf dem Objekt
 - > vorberechnen der Beleuchtung in 2D-Textur
 - > Textur heißt Environment-Map



Environment-Map

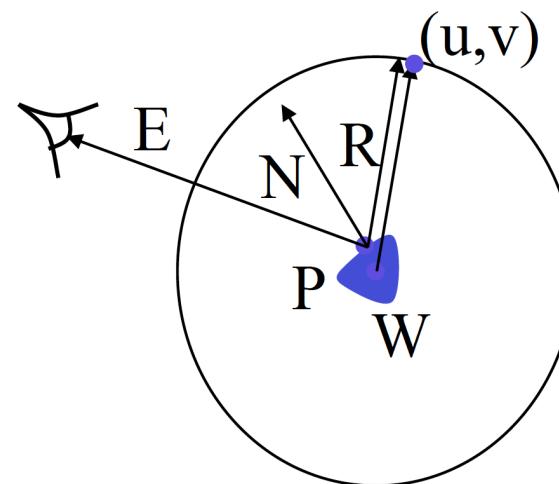


beleuchtete Szene



Environment-Mapping

- > Anschauung
 - > reflektierende Objekt wird von virtueller Kugel umgeben
 - > Alternative für Kugel: Würfel
 - > Environment-Map auf Innenseite der Kugel
 - > Texturkoordinaten ergeben sich aus Richtung des reflektierten Sehstrahls
 - > Berechnung des reflektieren Vektors
 - > $R = E - 2(E \bullet N)N$

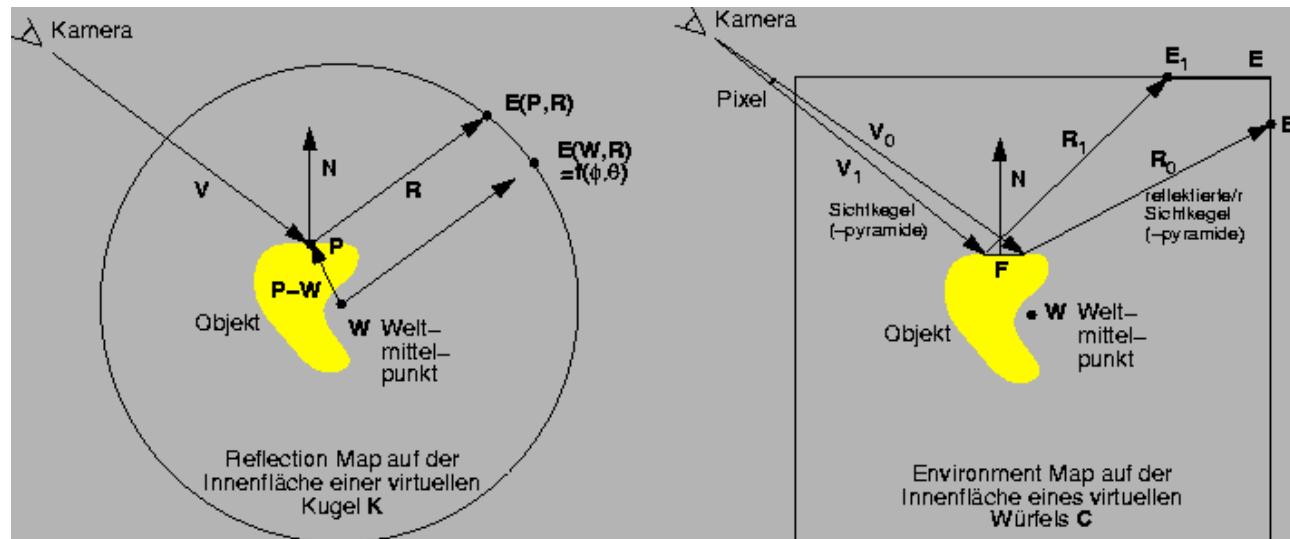


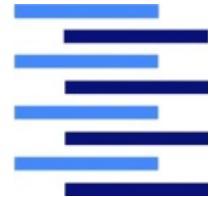


Environment-Mapping

Eigenschaften/Probleme

- > Reflexionsberechnung nur korrekt, wenn P im Weltmittelpunkt W liegt
- > keine Verdeckungsrechnung
- > keine gegenseitige Spiegelung von Szenenobjekten
- > Parametrisierung der Environment-Map





Environment-Mapping

- > Sphere-Mapping in OpenGL
 - > keine Kugelkoordinaten, sondern 6 Segmente auf der Kugel
 - > Segmente decken in Summe die Kugelfläche ab

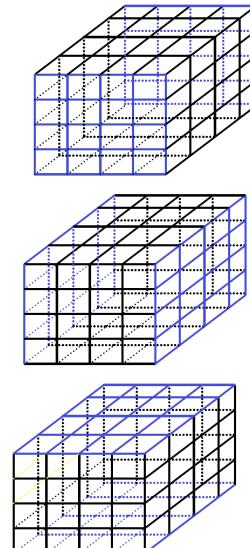
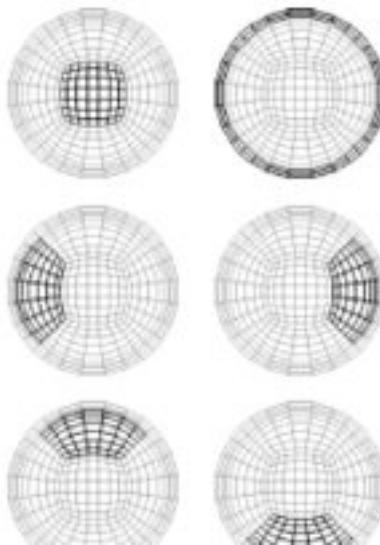


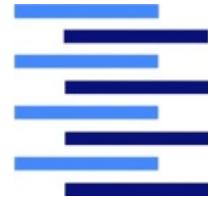
Abbildung der Texturen auf die Kugeloberfläche



reflektierende Kugel



Sphere-Maps



Bump-Mapping

- > Material und Farbe allein erzeugen oft zu glatte Oberflächen
- > realistische Oberflächen brauchen Struktur
- > Idee
 - > Modellierung der geometrischen Struktur als Offset-Fläche
 - > $P'(u, v) = P(u, v) + h(u, v) N(u, v) / |N(u, v)|$



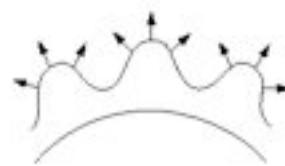
(a) Originalfläche $P(u)$
mit Normalen $N(u)$



(b) Bump Map $h(u)$



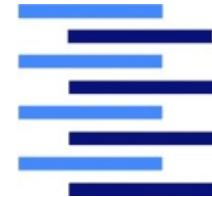
(c) Offsetfläche $P'(u)$



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



Oberfläche mit Bump-Map

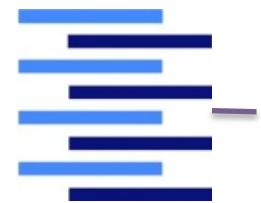


Bump-Mapping

- > Geometrische Positionen $P'(u, v)$ im Allgemeinen nicht notwendig
- > ausreichend, die perturbierten Normalen in der Beleuchtungsrechnung zu verwenden
 - > Gouraud-Shading nicht ausreichend
 - > Phong-Shading
 - > Umsetzung über Fragment-Shader



Oberfläche mit Bump-Map
(Quelle: [8])

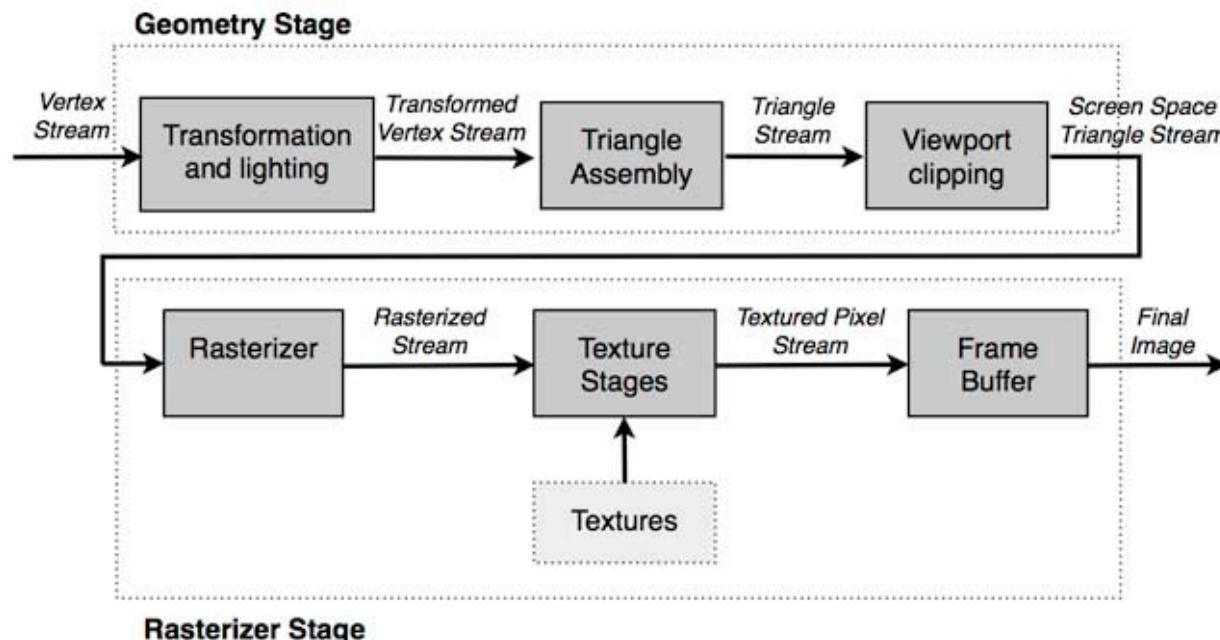


Shader



Shader

- > Traditioneller Ansatz
 - > Fixed Function Pipeline
 - > kaum Möglichkeit zur Einflussnahme

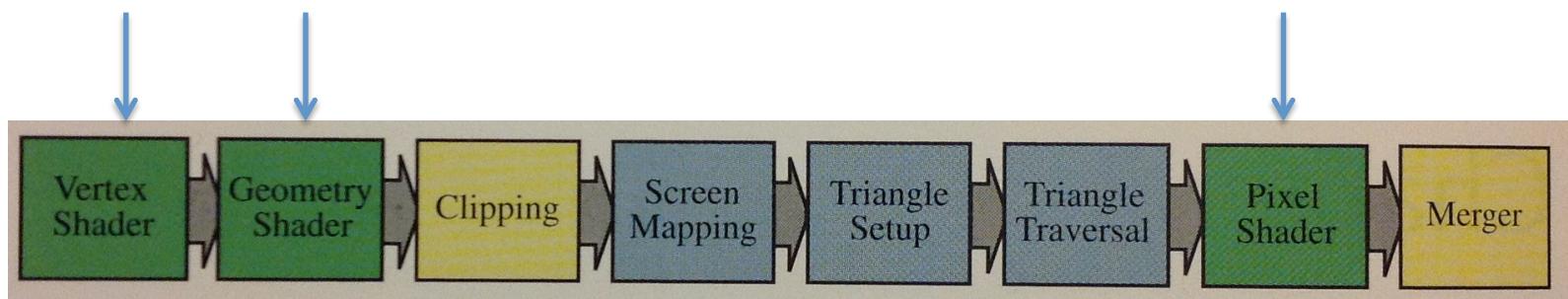


Fixed Function Pipeline (Quelle: [9])

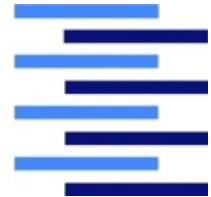


Shader

- > seit Shader Model 4, DirectX 10
 - > programmierbare Shader in der Grafik-Pipeline
 - > zentrale Komponenten
 - > Vertex Shader
 - > (Geometry Shader)
 - > Fragment oder Pixel Shader

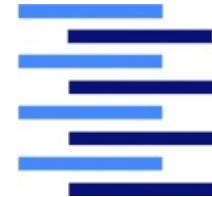


Programmierbare Grafik-Pipeline (Quelle: [10])



Shader

- > verschiedene Shader-Sprachen
 - > GLSL (OpenGL Shading Language): OpenGL
 - > HLSL (High Level Shading Language): DirectX
- > zusätzliche Hochsprachen zur vollständigen Programmierung/Ansteuerung von Grafikkarte
 - > CUDA (Compute Unified Device Architecture): nVidia-spezifisch
 - > OpenCL (Open Computing Language): Khronos-Gruppe (auch für OpenGL zuständig)
 - > DirectCompute: DirectX-basiert



Vertex Shader

- > Manipulation der Szenengeometrie
- > Verschieben der Vertices
- > Modifizieren der Form von Objekten
 - > nur bestehende Geometrien verändern
 - > keine neuen erzeugen oder löschen
- > Durchführen der Beleuchtungsrechnung basierend auf Vertices



Veränderungen an der Geometrie mit einem Vertex-Shader (Quelle: [10])



Fragment Shader (Pixel Shader)

- > Veränderung der zu rendernden Fragmente
- > z.B. realistischere Darstellung von Oberflächen- und Materialeigenschaften
- > Texturdarstellung verändern
- > Pixel-Farbwert kann aus mehreren Fragmenten zusammengesetzt werden
 - > z.B. Transparenz
- > Anwendungsfälle
 - > Phong Shading
 - > Spiegelungen
 - > Schattierung
 - > Lens Flares



Fragment-Shader für Cartoon-Effekt
(Quelle: [11])



Beispiel

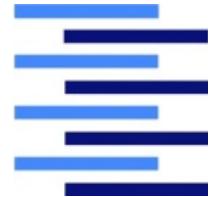
> Phong-Beleuchtungsmodell mit Gouraud-Shading – Vertex Shader (1/2)

```
varying vec4 color;
varying vec3 N;
varying vec3 v;

/**
 * Vertex shader used for Gouraud shading. Evaluate the lighting using
 * the Phong model at the vertices only.
 */
void main()
{
    // Define material properties.
    vec4 ambientMat = vec4(0,0,0,1);
    vec4 diffuseMat = vec4(0.25,0.75, 0.25,1);
    vec4 specMat = vec4(1,1,1,1);
    float specPow = 20.0;

    // Define required variables.
    vec4 diffuse;
    vec4 spec;
    vec4 ambient;

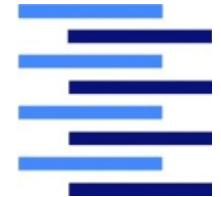
    ...
}
```



Beispiel

> Phong-Beleuchtungsmodell mit Gouraud-Shading – Vertex Shader (2/2)

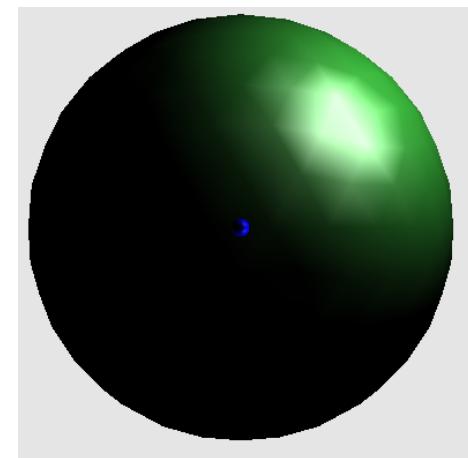
```
// Compute viewing direction.  
v = vec3(gl_ModelViewMatrix * gl_Vertex);  
// Compute normal direction.  
N = normalize(gl_NormalMatrix * gl_Normal);  
// Compute position in 3-space.  
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;  
  
// Normalize vectors.  
vec3 L = normalize(gl_LightSource[0].position.xyz - v);  
vec3 E = normalize(-v);  
vec3 R = normalize(reflect(-L,N));  
  
// Assemble color from the three basis components.  
ambient = ambientMat;  
diffuse = clamp( diffuseMat * max(dot(N,L), 0.0) , 0.0, 1.0 ) ;  
spec = clamp ( specMat * pow(max(dot(R,E),0.0),0.3*specPow) , 0.0, 1.0 ) ;  
color = ambient + diffuse + spec;  
}
```

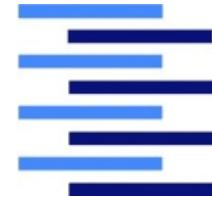


Beispiel

> Phong-Beleuchtungsmodell mit Gouraud-Shading – Fragment Shader

```
varying vec4 color;  
  
/**  
 * Fragment shader used for Gouraud shading. The vertex color is  
 * interpolated within the triangle.  
 */  
void main()  
{  
    // No operation required, just passing the color value from the vertex shader.  
    gl_FragColor = color;  
}
```





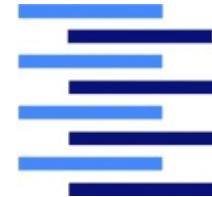
Beispiel

> Phong-Beleuchtungsmodell mit Phong-Shading – Vertex Shader

```
varying vec3 N;
varying vec3 v;

/**
 * Vertex shader used for Phong shading. No color computations are
 * required here, information is passed to the fragment shader.
 */
void main(void)
{
    // Compute the viewing direction.
    v = vec3(gl_ModelViewMatrix * gl_Vertex);
    // Compute the normal direction.
    N = normalize(gl_NormalMatrix * gl_Normal);
    // Compute the position in 3-space.
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;

}
```



Beispiel

> Phong-Beleuchtungsmodell mit Phong-Shading – Fragment Shader (1/2)

```
// These vectors need to be provided by the vertex shader
// Normal direction
varying vec3 N;
// Viewer direction
varying vec3 v;

/**
 * Fragment shader used for phong shading. The lighting color
 * value is computed for each pixel here.
 */
void main (void)
{
    // Define material
    vec4 ambientMat = vec4(0,0,0,1);
    vec4 diffuseMat = vec4(0.25,0.75, 0.25,1);
    vec4 specMat = vec4(1,1,1,1);
    float specPow = 20.0;

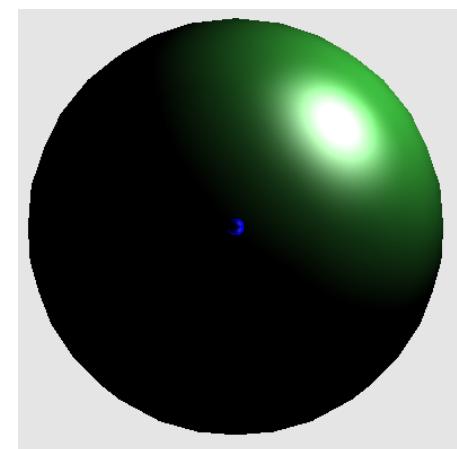
    // Define required variables.
    vec4 diffuse;
    vec4 spec;
    vec4 ambient;
```

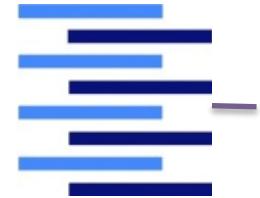


Beispiel

> Phong-Beleuchtungsmodell mit Phong-Shading – Fragment Shader (2/2)

```
// Evaluate the required vectors for the Phong lighting model.  
// Attention: Phong shading vs. Phong lighting  
// Only the first light source is considered.  
vec3 L = normalize(gl_LightSource[0].position.xyz - v);  
vec3 E = normalize(-v);  
vec3 R = normalize(reflect(-L,N));  
  
// Compute the color from the three basic components  
ambient = ambientMat;  
diffuse = clamp( diffuseMat * max(dot(N,L), 0.0) , 0.0, 1.0 ) ;  
spec = clamp ( specMat * pow(max(dot(R,E),0.0),0.3*specPow) , 0.0, 1.0 );  
gl_FragColor = ambient + diffuse + spec;  
}
```



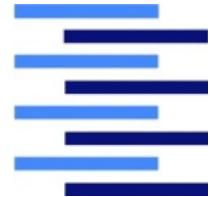


Zusammenfassung



Zusammenfassung

- > Texturen
 - > Einführung
 - > Interpolation
 - > Aliasing
 - > Erzeugen von Texturkoordinaten
 - > Weitere Anwendungen für Texturen
- > Shader



Quellen

- > Folien basieren auf den Vorlesungsfolien von Prof. Dr. Marc Alexa, Technische Universität Berlin, 2008
- > [1] <http://emileeknits.wordpress.com/category/swatch-sunday/>, abgerufen am 8.11.13
- > [2] <http://healthtoken.com/human-body-outline-for-kids-and-adult/human-body-outline-for-kids-459/>, abgerufen am 8.11.13
- > [3] http://cdn.overclock.net/f/ff/ffff1d246_2012-04-26_00003.jpeg, abgerufen am 8.11.13
- > [4] 123RF Bildnummer: 11812779, de.123rf.com, abgerufen am 8.11.13
- > [5] http://wiki.simigon.com/wiki/index.php?title=Terrain_texture_atlas, abgerufen am 8.11.13
- > [6] Alla Sheffer, Emil Praun and Kenneth Rose: Mesh Parameterization Methods and Their Applications, Foundations and Trends in Computer Graphics and Vision, Vol. 2, No 2 (2006) 105–171
- > [7] Quake I, ID Software
- > [8] http://www.chromesphere.com/tutorials/vue6/Optics_Basics_Print.html, abgerufen am 8.11.13
- > [9] <http://www.adobe.com/devnet/flashplayer/articles/how-stage3d-works.html>, abgerufen am 8.11.13
- > [10] T. Akenine-Möller, E. Haines, N. Hoffman: Real-Time Rendering, 3rd edition, CRC Press, 2008
- > [11] <http://prideout.net/blog/?cat=19>, abgerufen am 8.11.13