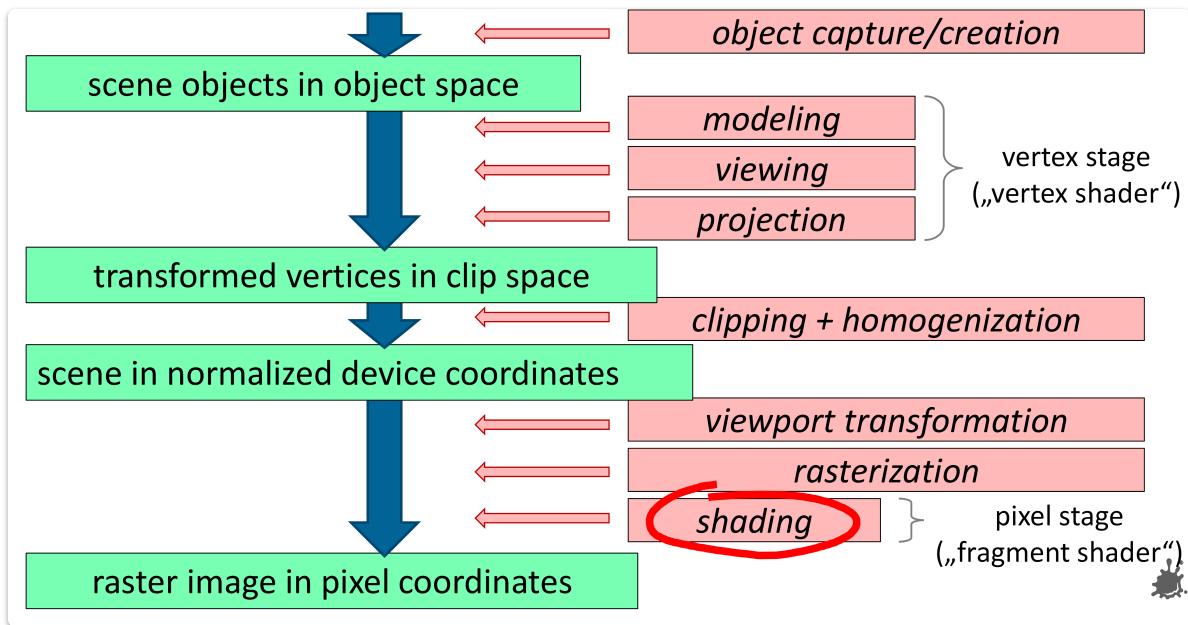


11. Globale Beleuchtung und Texturen

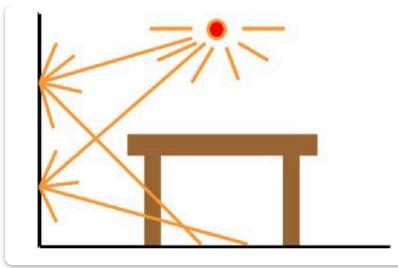
Rendering Pipeline



Radiosity

EVC_Skriptum_CG, p.43

- **Ursprung**: Wärmeübertragung.
- **Modellierung**: Lichtausbreitung unter Beachtung des Energiegleichgewichts in einem geschlossenen System.
- **Beschreibung**: Physikalischer Vorgang der Ausbreitung von Licht in einer diffus reflektierenden Umgebung.
- **Ziel**: Berechnung der Helligkeit aller Flächen einer Szene unter Berücksichtigung der gegenseitigen Beeinflussung.
- **Effekt**: Auch Flächen, die nicht direkt beleuchtet sind, erhalten eine gewisse Helligkeit (indirekte Beleuchtung).
- **Sekundäre Lichtquellen**: Jeder beleuchtete Gegenstand wirkt als sekundäre Lichtquelle und strahlt Licht in die Umgebung ab.
- **Bildgenerierung**:
 - Zuerst wird die Lichtausbreitung im Raum berechnet, **ohne** dass die Kameraposition bekannt ist.
 - Vereinfachende Annahme: Der Beobachter beeinflusst die Ausbreitung des Lichts nicht.
 - **Vorteil**: Objekte können dann aus verschiedenen Richtungen dargestellt werden, ohne dass die Lichtausbreitung jedes Mal neu berechnet werden muss.



Die Radiosity Gleichung

[EVC_Skriptum_CG, p.43](#)

- **Szenenbeschreibung:** Besteht aus n ebenen Polygonen, beim Radiosity-Verfahren als Patches bezeichnet.
- **Patch-Eigenschaften:**
 - Jedes Patch P_i ist homogen.
 - Perfekt diffuse Oberfläche (Licht wird in alle Richtungen gleichmäßig abgestrahlt).
 - Lichtquellen sind ebenfalls Patches.
- **Radiosity B_i von Patch P_i :**
 - Gesamte abgestrahlte Energie pro Flächeneinheit.
 - Summe aus Eigenemission und reflektierter Leistung pro Flächeneinheit.
 - Lichtenergiedichte ist proportional zur wahrgenommenen Helligkeit.
- **Vereinfachende Annahme:** Radiosity ist für alle Positionen auf einem Patch gleich.
- **Die Radiosity Gleichung:**

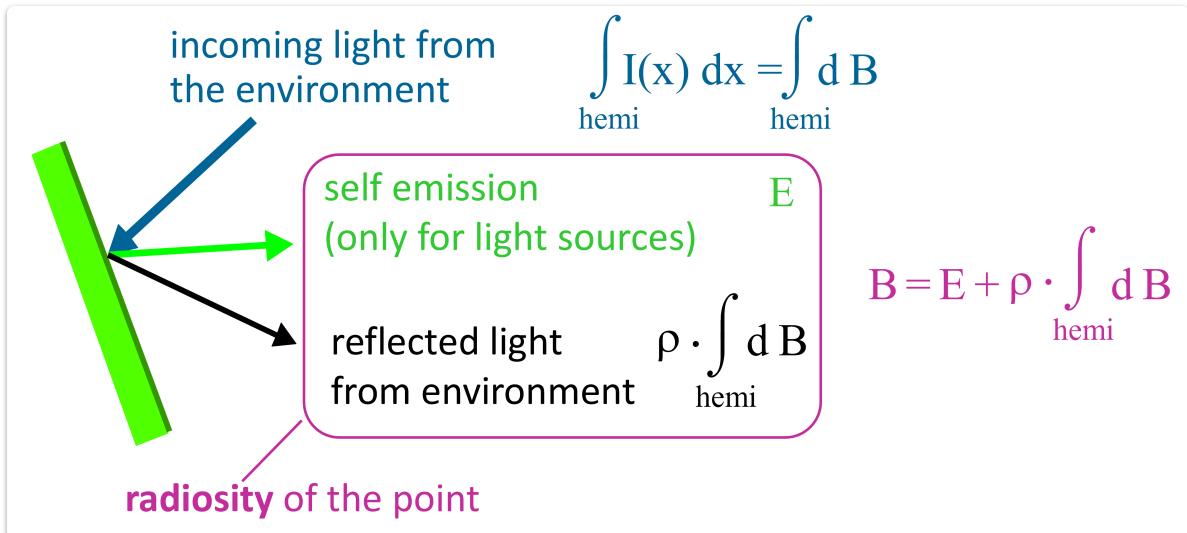
$$B_i = E_i + \rho_i \sum_{j=1}^n B_j F_{ij}$$

- B_i : Radiosity des Patches i .
- E_i : Eigenemission des Patches i .
- ρ_i : Diffuser Reflexionskoeffizient der Oberfläche von Patch i (gibt an, wie viel % des einfallenden Lichts diffus reflektiert wird, auch Albedo genannt).
- n : Anzahl der Patches in der Szene.
- B_j : Radiosity aller anderen Patches j .
- F_{ij} : Formfaktor (view factor) zwischen Patch i und Patch j .
 - Gibt an, welcher Anteil der Radiosity von Patch j auf Patch i wirkt (ist gleich dem Anteil der Radiosity von i , die auf j trifft, aufgrund des Reziprozitätsgesetzes).
 - Geometrische Größe, unabhängig von Lichtquellen oder Radiositywerten.
- **Gleichungssystem:** Die Radiosity-Formeln für n Patches ergeben ein lineares Gleichungssystem mit n Unbekannten B_i . (kann nur iterativ gelöst werden)
- **Matrix-Form des Gleichungssystems:**

$$B_i - \rho_i \sum_{j \neq i} B_j F_{ij} = E_i$$

$$\begin{pmatrix} 1 & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

Herleitung der Gleichung



to calculate the light influence between surfaces

Radiosity = total light leaving a surface point

$$B = E + \rho \cdot \int_{\text{hemi}} d B$$

B ... radiosity hemi ... half space over point

E ... self emission ρ ... reflection coefficient

“radiosity = self emission + reflection property · sum of all incoming light”

- diffuse interreflections in a scene
- radiant energy transfers
- conservation of energy, closed environments
- subdivision of scene into *patches* with constant radiosity B_i

$$B = E + \rho \cdot \int_{\text{hemi}} d B$$

the scene is discretized into **n "patches"** (plane polygons) P_i , for each of these patches a constant radiosity B_i is assumed:

$$B = E + \rho \cdot \int_{\text{hemi}} d B \quad \Rightarrow \quad B_i = E_i + \rho_i \cdot \sum_{j=1}^n B_j \cdot F_{ij}$$

ρ_i diffuse reflection coefficient of patch **i**

F_{ij} “form factor”: describes what % of the influence on patch **i** comes from patch **j**;
= geometric size !

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

B_i ... radiosity of patch **i**

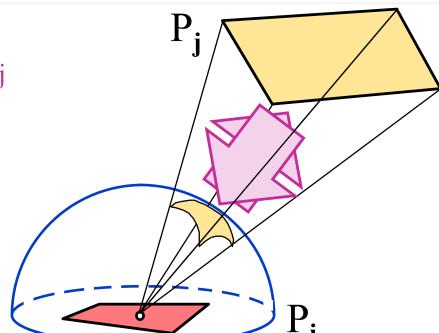
E_i ... self-emission of patch **i**

$\sum B_j F_{ij}$... contribution of other patches

F_{ij} ... form factor, defines

- contribution of B_i on patch **j** - which is equal to
- contribution of patch **j** to B_i

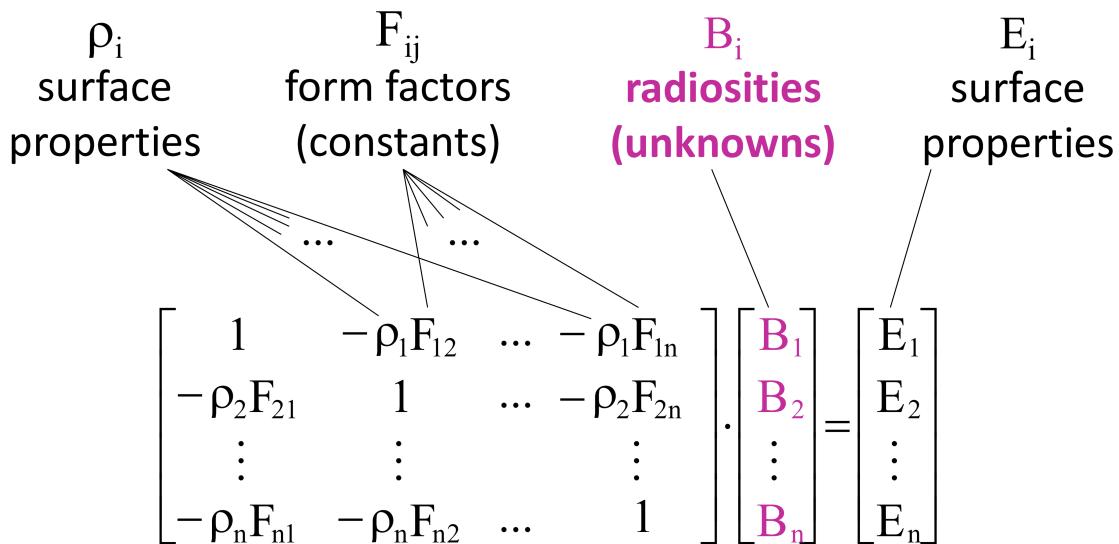
ρ_i ... reflectivity coefficient of patch **i** (“*albedo*”)



$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

$$B_i - \rho_i \sum_{j \neq i} B_j F_{ij} = E_i$$

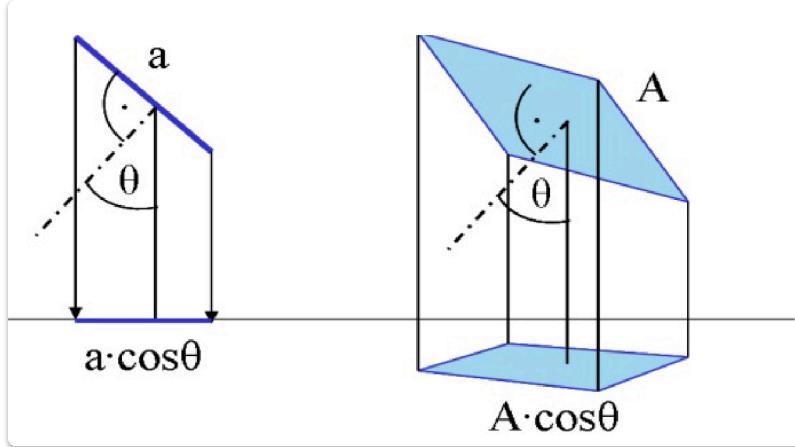
$$\begin{bmatrix} 1 & -\rho_1 F_{12} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 & \dots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$



Berechnung der Formfaktoren

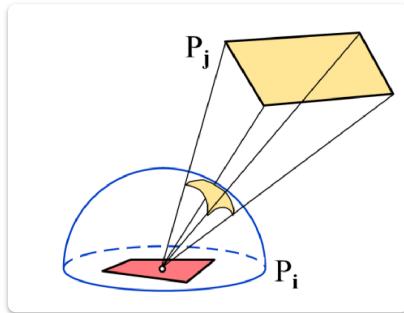
[EVC_Skriptum_CG, p.43, p.44](#)

- **Geometrischer Zusammenhang:** Die Fläche der Normalprojektion einer Fläche A auf eine andere Fläche verkleinert sich um den Kosinus des Winkels θ zwischen den Flächen: $A \cdot \cos \theta$.



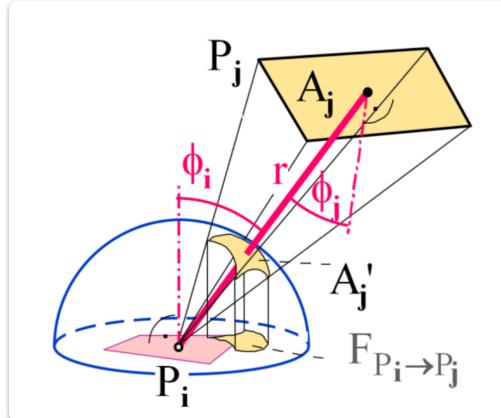
- **Definition des Formfaktors F_{ij} :**

- Anteil der vom Patch P_j ausgehenden Strahlungsenergie, die den Patch P_i trifft.
- Oder: Wie viel Prozent der Energie von P_j trifft auf P_i ?
- Reziprok: Gibt auch an, wie viel Prozent der auf P_i eingehenden Energie von P_j kommt.



- **Herleitung der Formel für F_{ij} (vereinfachte Annahme: Patches klein im Verhältnis zum Abstand r):**

1. Betrachte Patch P_i mit Fläche A_i .
2. Stelle dir über P_i eine Halbkugel (Hemisphäre) mit Radius 1 vor, auf die P_j projiziert wird.
3. Die projizierte Fläche A'_j hat die Größe $A_j \cos \phi_j$, wobei ϕ_j der Winkel zwischen der Normalen von P_j und der Verbindungsgeraden zwischen den Patches ist.



4. Energie, die unter einem Winkel ϕ_i auf P_i einfällt, ist proportional zu $\cos \phi_i$. Multiplikation mit $\cos \phi_i$ (entspricht einer Projektion auf die Grundfläche der Halbkugel) liefert den korrekten Anteil des Einflusses von Patch P_j auf P_i .
5. Normierung durch die Größe der Grundfläche der Halbkugel ($1^2\pi = \pi$).

- **Formel für den Formfaktor F_{ij} :**

$$F_{ij} = \frac{\cos \phi_i \cos \phi_j A_j}{\pi r^2}$$

- **Wichtige Voraussetzung:** Diese Formel gilt unter der Annahme, dass sich keine Hindernisse zwischen den beiden Patches befinden und das Licht ungehindert von P_j nach P_i gelangen kann. Korrekte Formfaktoren müssen auch die gegenseitige Sichtbarkeit berücksichtigen.
- ϕ_i : Winkel zwischen der Normalen von P_i und der Verbindungsgeraden.
- ϕ_j : Winkel zwischen der Normalen von P_j und der Verbindungsgeraden.
- A_j : Fläche des Patches P_j .
- r : Abstand zwischen den Patches.
- **Reziprozitätsprinzip:** Die Abhängigkeit der Formfaktoren zwischen zwei Patches setzt in Beziehung:

$$A_i F_{ij} = A_j F_{ji}$$

form factor F_{ij} : contribution of patch P_j to B_i = contribution of B_i to patch P_j

form factor properties:

■ conservation of energy

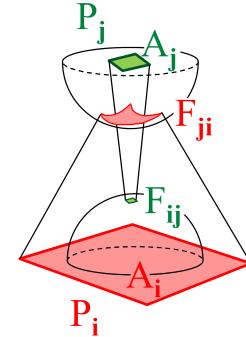
$$\sum_{j=1}^n F_{ij} = 1$$

■ uniform light reflection

$$A_i F_{ij} = A_j F_{ji}$$

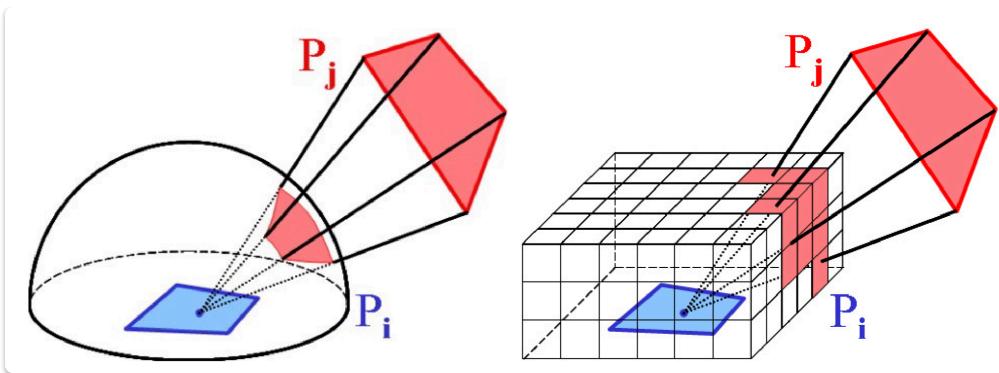
■ no self-incidence

$$F_{ii} = 0$$



Praktische Berechnung:

- **Hemicube-Methode**
 - Abschätzung der Formfaktoren statt einer Halbkugel über P_i wird ein Halbwürfel (Hemicube) verwendet.
 - Die gesamte Szene wird auf diesen Hemicube abgebildet (z-Buffer-Technologie).
 - Die Oberfläche des Halbwürfels wird in Pixel aufgeteilt.
 - Alle anderen Patches werden von P_i aus mit dem Würfelmittelpunkt als Projektionszentrum auf diese Pixel abgebildet.
 - Für jedes Pixel wird sein Formfaktor bestimmt.
 - Für jedes Patch wird die Summe der Formfaktanteile seiner Pixel gebildet.
- **Ray-Tracing-Verfahren:** Alternative Methode zur Berechnung von Formfaktoren.



Fortschreitende Verfeinerung (Progressive Refinement)

EVC_Skriptum_CG, p.44

$$B_i = E_i + \rho_i \sum_{j \neq i} B_j F_{ij}$$

"shooting" → select brightest patch P_i and distribute its radiosity B_i

$$B_{j \text{ due to } B_i} = \rho_j B_i F_{ij} \frac{A_i}{A_j}$$

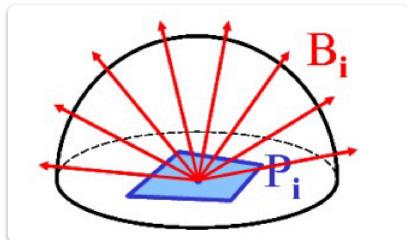
- **Ziel:** Iterative Lösung des Radiosity-Gleichungssystems.
- **Grundidee ("Shooting"):** Man wählt das jeweils hellste Patch aus und "schießt" (verteilt) dessen noch nicht abgestrahlte Energie auf alle anderen Patches.
- **Vorteil:**
 - In jedem Schritt werden alle Patches etwas besser beleuchtet.
 - Das Verfahren konvergiert schnell, da immer die Energie der hellsten Patches zuerst verteilt wird.
- **Radiosity-Anteil von P_i , der von P_j verursacht wird ($B_{(i \text{ von } B_j)}$):**
 - Dies beschreibt, wie viel Radiosity von Patch j zu Patch i beiträgt.
 - Es gilt: $B_{(i \text{ von } B_j)} = \rho_i B_j F_{ij}$
 - **Symmetrie:** Der Einfluss von P_i auf P_j ist symmetrisch zum Einfluss von P_j auf P_i .
 - Es gilt auch: $B_{(j \text{ von } B_i)} = \rho_j B_i F_{ji}$
- **Verknüpfung mit dem Reziprozitätsprinzip:**
 - Aus dem Reziprozitätsprinzip $A_i F_{ij} = A_j F_{ji}$ folgt, dass $F_{ij} = F_{ji} \frac{A_j}{A_i}$.
 - Setzt man dies in die Gleichung für $B_{(i \text{ von } B_j)}$ ein:

$$B_{(i \text{ von } B_j)} = \rho_i B_j F_{ij} = \rho_i B_j F_{ji} \frac{A_j}{A_i}$$
 - Diese Beziehung ermöglicht es, den Formfaktor F_{ij} direkt zu nutzen, auch wenn die Berechnung des Beitrags von P_j zu P_i erfolgt.
- **Speicherhaltung pro Patch:**

- **Gesammelte Radiosity (B_i):** Der bisher beste Schätzwert für die Radiosity des Patches.
- **Noch nicht verschossene Radiosity (ΔB_i):** Die Energie, die dieses Patch noch abgeben muss und die als Basis für die Auswahl des nächsten "hellsten" Patches dient.
- **Initialisierung:**

- Am Anfang werden B_i und ΔB_i mit der Eigenemission E_i des Patches initialisiert:

$$B_i = \Delta B_i = E_i \text{ für alle Patches } i.$$



Iterationsschritt des Progressive Refinement Algorithmus

p.45

```

select patch  $i$  with highest  $A_i * \Delta B_i$ 
FOR selected patch  $i$  {
    set up hemicube
    calculate form factors  $F_{ij}$ 
}
FOR each patch  $j$  {
     $\Delta rad := \rho_j * \Delta B_i * F_{ij} * A_i / A_j$ 
     $\Delta B_j := \Delta B_j + \Delta rad$ 
     $B_j := B_j + \Delta rad$ 
}
 $\Delta B_i := 0$ 

```

- **Bezeichnung:** Dieses Verfahren wird auch als "Progressive Refinement" (schrittweise Verfeinerung) bezeichnet.

Verschiedene Beispiele: [EVC-CG11-Globale Beleuchtung+Texturen_2025S_Slides](#), p.19

Aspekte von Radiosity

[EVC_Skriptum\(CG\)](#), p.45

- **Blickpunktunabhängigkeit:** Radiosity ist eine Methode zur Berechnung der Helligkeit von einzelnen diffusen Patches, die unabhängig von der Kameraposition (Blickpunkt) ist.
 - **Vorteil:** Die Lichtausbreitung muss nur einmal berechnet werden, danach können Ansichten aus beliebigen Blickpunkten generiert werden, ohne die Beleuchtung neu zu berechnen.

- **Nachfolgender Rendering-Schritt:** Nach der Radiosity-Berechnung ist meist noch ein zusätzlicher Rendering-Schritt notwendig, um das finale Bild zu erzeugen.
 - Oft wird hierfür ein einfaches Polygon-Verfahren mit Gouraud-Schattierung verwendet, um die berechneten Helligkeiten darzustellen.
- **Kombination mit Ray-Tracing:** Die durch Radiosity gewonnenen diffusen Schattierungswerte können als Basiswerte für ein Ray-Tracing-Verfahren verwendet werden.
 - **Vorteil:** Dadurch lassen sich zusätzlich Effekte wie Spiegelungen und scharfe Schatten, die Radiosity allein nicht modelliert, gut darstellen.

radiosity is viewpoint-independent
needs a rendering step to display

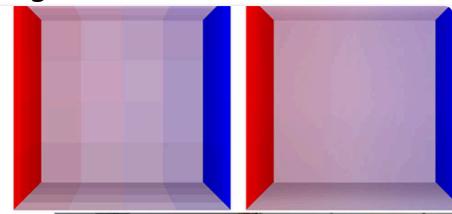
- polygon rendering
- Gouraud shading
- ray-tracing
- ...

combination with ray-tracing enables

- reflections
- shadows
- ...

Werner Purgathofer

23



© H. Wann Jense

Erweiterungen des Radiosity-Prinzips

EVC_Skriptum_CG, p.45

Das vorgestellte Radiosity-Prinzip ist erweiterbar, um Effizienz und Qualität zu verbessern:

- **Hierarchische Strukturierung von Patches:**
 - **Ziel:** Reduzierung der Anzahl der zu berechnenden Patches.
 - **Methode:** Patches können hierarchisch strukturiert werden. Das bedeutet, dass entfernte Patches nicht einzeln behandelt werden müssen, sondern als Gruppen zusammengefasst werden können. Dies reduziert den Rechenaufwand erheblich.
- **Discontinuity-Meshing:**
 - **Problem:** An Stellen, an denen sich die Beleuchtung abrupt ändert (z.B. Schattenkanten), können zu große Patches eine falsche "Verschmierung" der Beleuchtung verursachen. Umgekehrt können zu kleine Patches den Rechenaufwand unnötig erhöhen.
 - **Lösung:** Discontinuity-Meshing passt die Größe der Patches an die Beleuchtungssituation an. In Bereichen mit starken Helligkeitsunterschieden (z.B. Schattenübergängen) werden kleinere Patches verwendet, um eine präzisere Darstellung zu ermöglichen. In Bereichen mit gleichmäßiger Beleuchtung können größere Patches verwendet werden.

Path Tracing

EVC_Skriptum_CG, p.45

- **Grundlage:** Path Tracing ist eine Erweiterung des [Ray-Tracing Verfahrens](#).
- **Alternativname:** Wird auch "*Monte Carlo Ray-Tracing*" genannt.
- **Kernprinzip:**
 - Beim klassischen Ray-Tracing werden an jedem Auftreffpunkt Sekundärstrahlen in alle relevanten Richtungen gelegt (z.B. Reflexion, Brechung, Schatten).
 - Beim Path Tracing wird stattdessen an jedem Auftreffpunkt **zufällig nur eine Richtung** entsprechend einer gültigen Verteilungsfunktion ausgewählt.
 - **Intuition:** Es wird ein "Lichtpfad" (Path) von der Kamera bis zu einer Lichtquelle oder ins Unendliche verfolgt, wobei an jeder Oberfläche nur eine zufällige Richtung für die Weiterverfolgung gewählt wird.
- **Vorteile:**
 - **Inklusion komplexer Lichtverhältnisse:** Ermöglicht die realistische Simulation von Szenen, in denen viele verschiedene Lichtrichtungen relevant sind.
 - **Diffuse Reflexion:** Kann diffuse Reflexionen sehr gut modellieren (im Gegensatz zu klassischem Ray-Tracing, das eher auf spiegelnde oder brechende Oberflächen spezialisiert ist).
 - **Ausgedehnte Lichtquellen:** Kommt besser mit ausgedehnten (nicht punktförmigen) Lichtquellen zurecht.
- **Herausforderungen & Lösungen:**
 - **Rauschen im Ergebnisbild:** Da pro Pixel nur eine zufällige Richtung verfolgt wird, führt dies zu Rauschen im Ergebnis.
 - **Lösung:** Um starkes Rauschen zu reduzieren, müssen pro Pixel **viele Strahlen** (Pfade) berechnet und gemittelt werden. Dies ist rechenintensiv.
- **Mathematischer Hintergrund:**
 - Grundsätzlich entspricht das Vorgehen des Path Tracings der [Monte-Carlo-Integration](#) eines mehrdimensionalen Integrals.
 - Dieses Integral wird als "Rendering Equation" bezeichnet und beschreibt die vollständige Lichtausbreitung im Raum.
 - **Rendering Equation (vereinfachtes Verständnis):** Eine komplexe mathematische Gleichung, die die gesamte Beleuchtung eines Punktes in einer Szene beschreibt, indem sie alle eingehenden Lichtstrahlen und deren Wechselwirkungen mit der Oberfläche berücksichtigt. Monte Carlo Integration ist eine Methode, um solche komplexen Integrale durch zufällige Stichproben zu approximieren.

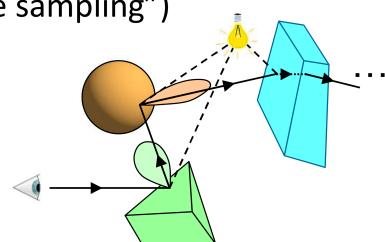
also called Monte Carlo ray tracing

ray directions selected randomly according to

distribution functions (“importance sampling”)

uses Monte Carlo integration to solve

$$B = E + \rho \cdot \int_{\text{hemi}} d B$$



B ... radiosity **hemi** ... half space over point

E ... self emission **ρ** ... reflection coefficient

- **Qualitätsverbesserung:**

- Die Verwendung von **Quasi-Zufallszahlen** (anstelle von Pseudo-Zufallszahlen) reduziert die Varianz (d.h., das Rauschen) im Ergebnisbild merklich.
- **Quasi-Zufallszahlen:** Sind keine echten Zufallszahlen, sondern Sequenzen, die eine gleichmäßige Verteilung im Definitionsbereich aufweisen als Pseudo-Zufallszahlen. Dies hilft, die Stichproben für die Monte-Carlo-Integration besser über den gesamten Raum zu verteilen und somit das Ergebnis genauer zu machen.

Photon Mapping

[EVC_Skriptum_CG](#), p.46

- **Grundidee:** Eine Methode zur Berechnung globaler Beleuchtungseffekte, die besonders gut mit komplexen Lichtinteraktionen wie Spiegelungen und Kaustiken (Lichtbündel, die durch Brechung oder Reflexion erzeugt werden) umgehen kann.

- **Vorgehensweise:**

1. **“Vorwärts”-Verfolgung von Lichtstrahlen:** Im Gegensatz zum Ray-Tracing, das Strahlen von der Kamera verfolgt, werden beim Photon Mapping Lichtstrahlen (genannt “Photonen”) von den Lichtquellen ausgesendet und in der Szene verfolgt. Dies ist eine “Vorwärts”-Richtung.
2. **Speichern der Lichtwirkung an Auftreffpunkten:** Wenn ein Photon auf eine Oberfläche trifft, wird die Wirkung des Lichts (z.B. Energie, Farbe) an diesem Auftreffpunkt gespeichert. Diese gespeicherten Punkte werden als “Photonen” in einer “Photon Map” abgelegt.
3. **Interpolation für das Aussehen des Objekts:** Später, während des Renderings, werden diese gespeicherten Photonen genutzt. An einem Punkt, für den die Beleuchtung berechnet werden soll, werden die Informationen von mehreren nahegelegenen Photonen interpoliert, um das Aussehen des Objekts zu bestimmen.

- **Vorteile:**

- **Korrekte Berechnung komplexer Lichtsituationen:** Ermöglicht die akkurate Simulation von:
 - **Spiegelungen der Lichtquellen:** Wie sich Lichtquellen in spiegelnden

Oberflächen abbilden.

- **Kaustiken:** Die Fokussierung von Licht durch Brechung (z.B. durch Glas) oder Reflexion (z.B. durch eine gewölbte, spiegelnde Oberfläche), die zu hellen Lichtmustern auf anderen Oberflächen führt.
- **Kombination mit anderen Verfahren:**
 - Eine Kombination aus **Path Tracing** und **Photon Mapping** kann nahezu alle Lichteffekte in einem Bild integrieren und so sehr realistische Renderings erzeugen. Path Tracing ist gut für direkte und diffuse Beleuchtung, während Photon Mapping seine Stärken bei Kaustiken und komplexen spiegelnden/refraktiven Pfaden hat.

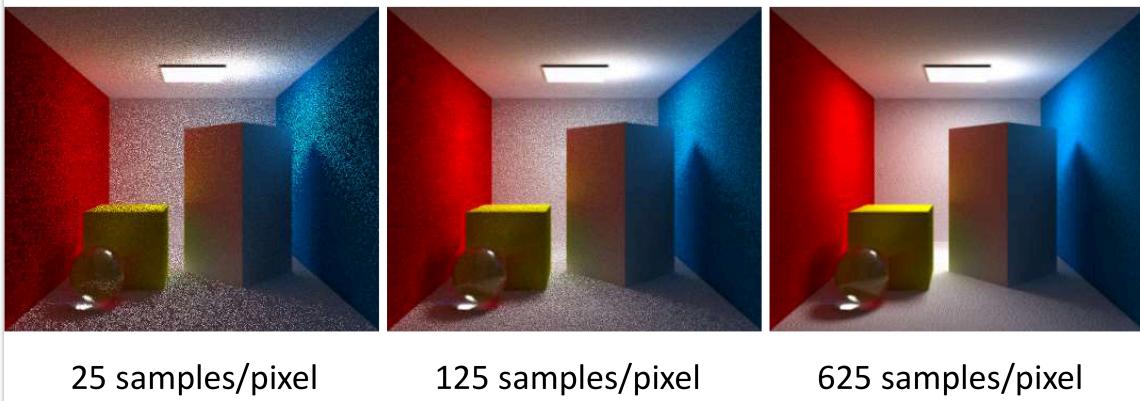


Optische Eigenschaften natürlicher Oberflächen: Mapping-Methoden

EVC_Skriptum_CG, p.46

- **Problem:** Natürliche Oberflächen weisen diverse Unregelmäßigkeiten auf, die entweder durch die Umgebung, eine variable Färbung der Oberfläche oder durch Oberflächenunebenheiten (Schattierung) verursacht werden.
- **Simulationsmethoden ("Mapping"):** Für diese drei Ursachen lassen sich die Effekte mit speziellen "Mapping"-Methoden simulieren:
 - **Environment Mapping:** Simuliert die Reflexion der Umgebung auf einer Oberfläche. (Beispiel: Eine glänzende Kugel spiegelt die Umgebung wider, ohne dass die Umgebung geometrisch modelliert werden muss.)
 - **Texture Mapping:** Bringt ein 2D-Bild (Textur) auf eine 3D-Oberfläche auf, um deren Farbe und Muster zu definieren. (Beispiel: Eine Ziegelmauertextur auf einer Wand.)
 - **Bump Mapping:** Simuliert Oberflächenunebenheiten, indem es die Normalenvektoren der Oberfläche modifiziert. Dies beeinflusst die Schattierung und erzeugt den Eindruck von Unebenheiten, ohne die Geometrie tatsächlich zu verändern. (Beispiel: Eine grobe Steintextur, die Risse und Vertiefungen simuliert.)
- **"Mapping" Bedeutung:** Im Kontext der Computergrafik bedeutet "Mapping" das Abbilden von Informationen (wie Farben, Helligkeiten, Normalen) von einem Raum (oft 2D-Bild) auf eine 3D-Oberfläche.

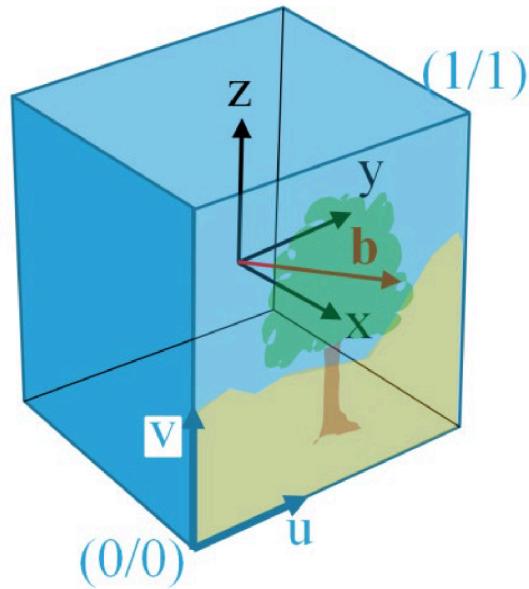
→ trace light rays from light source(s) and store illumination on objects



Environment Mapping

[EVC_Skriptum_CG, p.46](#)

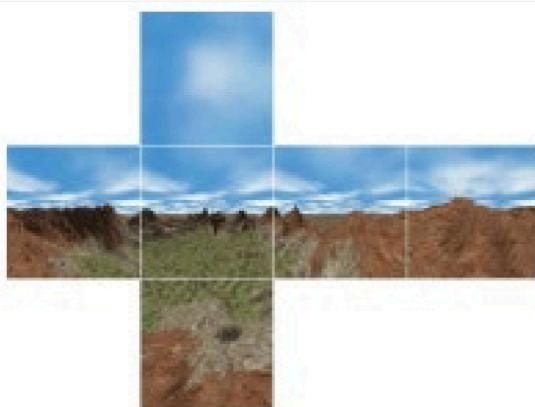
- **Definition:** Environment Mapping (Umgebungsabbildung) bezeichnet die Technik, wie die Umgebung oder Umwelt eines Objekts dessen Aussehen beeinflusst, insbesondere durch Reflexionen.
- **Effekt:** Je nach Oberflächeneigenschaften des Objekts wird die Umgebung auf verschiedene Weisen im Erscheinungsbild des Objekts wiedergespiegelt.
- **Anwendung bei verschiedenen Oberflächen:**
 - **Perfekt spiegelnde Oberflächen:** Für diese Oberflächen würde Ray-Tracing das exakte Spiegelbild der Umgebung erzeugen. Environment Mapping kann dies annähernd simulieren.
 - **Nicht perfekt diffuse Oberflächen:** Für diese Oberflächen, die einen gewissen Glanz aufweisen, kann man mit dem Phong-Modell (oder ähnlichen Reflexionsmodellen) einen Glanz-Effekt erzeugen, der die Spiegelung von Lichtquellen annähert.
 - **Mehr oder weniger spiegelnde Oberflächen:** Reflektieren ihre ganze Umgebung mehr oder weniger scharf. Die Genauigkeit der Spiegelung wird dabei reduziert, d.h. sie ist nicht exakt.
- **Motivation für Environment Mapping:**
 - Um ein effizientes Rendering von Objekten in einer komplexen Umgebung zu ermöglichen, ohne die gesamte Umgebung vollständig und exakt modellieren zu müssen.
 - **Vorgehen:** Die Umgebung wird in einem Vorverarbeitungsschritt von einem zentralen Punkt aus (z.B. dem Mittelpunkt des Objekts oder der darzustellenden Szene) als Bild (oder Satz von Bildern) produziert.



- **Art der Umgebungsinformation:**

- Es spielt keine Rolle, ob die Umgebungsbilder berechnet oder fotografiert wurden.
- **Wichtig:** Die Umgebung ist im Verhältnis zu den Objekten, die man darstellen will, sehr groß.
- **Annahme:** Bei der Darstellung eines Objekts wird für jeden Oberflächenpunkt näherungsweise angenommen, dass er sich im Zentrum dieser Umgebung befindet. Dies ist eine Vereinfachung, die für Objekte, die klein im Vergleich zur Umgebung sind, gute Ergebnisse liefert.

- **Vorteil:** Man kann allein aus der Richtung des Reflexionsstrahls bestimmen, welcher Umgebungschnitt getroffen wird (z.B. mit Polarkoordinaten), und erspart sich aufwändiges Ray-Tracing für die Umgebungsinformation.

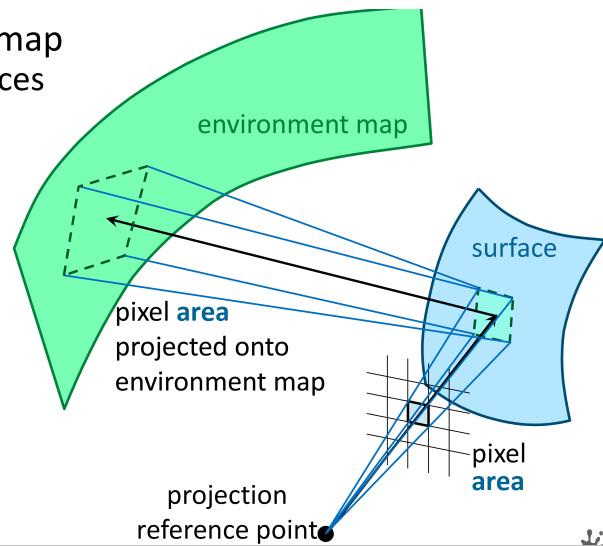


information in the environment map

- intensity values for light sources
- sky
- background objects

pixel area

- projected onto surface
- reflected onto environment map



Cube Map als Environment Map

EVC_Skriptum_CG, p.46

- **Häufigste Methode:** Häufig wird ein achsenparalleler Würfel als Environment Map verwendet, der als **Cube Map** bezeichnet wird.
- **Struktur der Cube Map:** Jede Seite dieses Würfels ist ein Bild der Größe $u \times v = [0, 1] \times [0, 1]$.
- **Effiziente Bildwertabfrage:** Um den Bildwert für eine bestimmte Reflexionsrichtung R_S effizient zu bestimmen, kann man die x, y, z Koordinaten des Reflexionsvektors nutzen.
 - **Beispiel (für die Seite $+x$):** Wenn der dominante Anteil des Reflexionsvektors die x -Komponente ist, wird die Textur der $+x$ -Seite verwendet. Die u, v -Koordinaten werden dann aus den verbleibenden Komponenten y und z abgeleitet:
 - $u = (y_R + x_R)/(2x_R)$
 - $v = (z_R + x_R)/(2x_R)$
 - Hierbei ist $R_S = (x_R, y_R, z_R)$ der Reflexionsvektor.
- **Reflexionsrichtung R_S :** Die Reflexionsrichtung erhält man nach dem gleichen Prinzip wie schon beim Ray-Tracing und bei den Schattierungsverfahren abgeleitet:
 - $R_S = (2n \cdot v)n - v$
 - v : Vektor vom Betrachter zum Oberflächenpunkt.
 - n : Oberflächennormale am Punkt.
 - **Intuitive Erklärung:** Dies ist die klassische Formel für die Reflexion eines Vektors an einer Oberfläche. Der Term $(2n \cdot v)n$ ist der Anteil des Vektors v , der parallel zur Normalen n ist, gespiegelt an der Oberfläche. Davon wird der ursprüngliche Vektor v subtrahiert, um die reine Reflexionsrichtung zu erhalten.



(1/1) find (u, v) from the direction vector $\mathbf{r}(x_r, y_r, z_r)$:

if $x_r > |y_r|$ and $x_r > |z_r|$ then "front face"

$$u = (y_r + x_r)/2x_r$$

$$v = (z_r + x_r)/2x_r$$

top view

analogous formulas for the other 5 faces
 $(-x > |y| \wedge -x > |z|, \quad y > |x| \wedge y > |z|, \quad -y > |x| \wedge -y > |z|, \quad z > |x| \wedge z > |y|, \quad -z > |x| \wedge -z > |y|)$

(1/1) calculation of the direction vector \mathbf{r} :

for a pixel: viewing direction \mathbf{v} and normal vector $\mathbf{n} \Rightarrow \mathbf{r} + \mathbf{v} = (2\mathbf{n} \cdot \mathbf{v})\mathbf{n}$

$$\mathbf{r} = (2\mathbf{n} \cdot \mathbf{v})\mathbf{n} - \mathbf{v}$$

Texture Mapping

[EVC Skriptum CG, p.47](#)

- Problem:** Viele Oberflächen in der realen Welt sind nicht einfarbig, sondern weisen komplexe Muster, Farben und Details auf (z.B. Holzmaserung, Bilder, Schrift, Verschmutzung, Kleidung, Marmor).
- Lösung:** Um solche Muster auf Oberflächen darzustellen, auch wenn die grundlegende geometrische Modellierung grob ist, wird die Technik des **Texture Mapping** verwendet.
- Was ist eine Textur?** Ein Muster, das auf eine Oberfläche aufgebracht wird. Beispiele: Fenster auf einer Hauswand, Wolken am Himmel, Gesichter, Knöpfe, Reißverschlüsse, Pflastersteine.

- **Zweischrittiger Prozess des Texture Mapping:**

1. **Textur-Objekt Transformation (Modellierungsschritt):**

- **Definition:** Zuerst muss definiert werden, welche Textur (oft ein 2D-Bild) auf welche Oberfläche des 3D-Objekts aufgebracht werden soll.
- **Parameter:** Dabei werden Orientierung, Skalierung, Wiederholung (Tiling) und andere Parameter festgelegt.
- **Raum:** Hier findet eine Transformation von **Textur-Raum (u,v) Array Koordinaten** (die 2D-Koordinaten innerhalb der Textur) in **Objekt-Raum (u',v')** **Flächen-Parameter** (Parameter, die einen Punkt auf der 3D-Oberfläche des Objekts eindeutig identifizieren) statt.
- **Bedeutung:** Dieser Schritt ist eigentlich Teil der Modellierung, bei der die Oberflächen ein visuelles Aussehen erhalten.

2. **Viewing- & Projektions-Transformation (Rendering-Schritt):**

Ziel: Die Textur muss korrekt auf das Abbild des Objekts im finalen Bild gerendert werden.

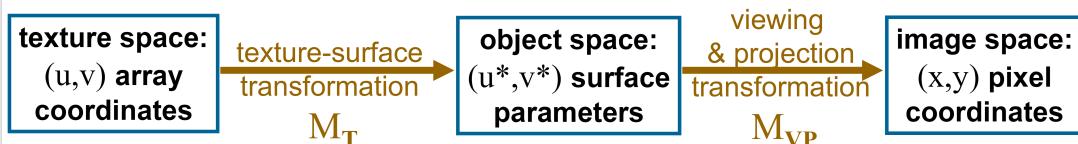
Raum: Dies beinhaltet die Transformation von den **Objekt-Raum (u',v')** **Flächen-Parametern** in **Bild-Raum (x,y) Pixel Koordinaten**.

* **Bedeutung:** Hier wird die Textur tatsächlich auf das gerenderte Objekt projiziert und dargestellt.

texture mapping

forward: texture scanning $(u,v) \rightarrow (x,y)$

backward: inverse scanning $(x,y) \rightarrow (u,v)$

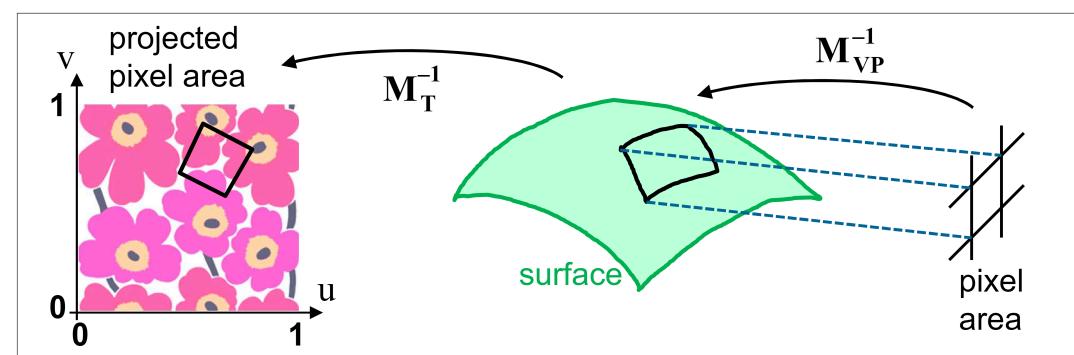


texture-surface transformation

$$\begin{aligned} u^* &= u^*(u,v) = a_{u^*}u + b_{u^*}v + c_{u^*} \\ v^* &= v^*(u,v) = a_{v^*}u + b_{v^*}v + c_{v^*} \end{aligned}$$

projecting pixel areas to texture space =

= inverse scanning $(x,y) \rightarrow (u,v)$



Erzeugung einer Textur

[EVC_Skriptum_CG, p.47](#)

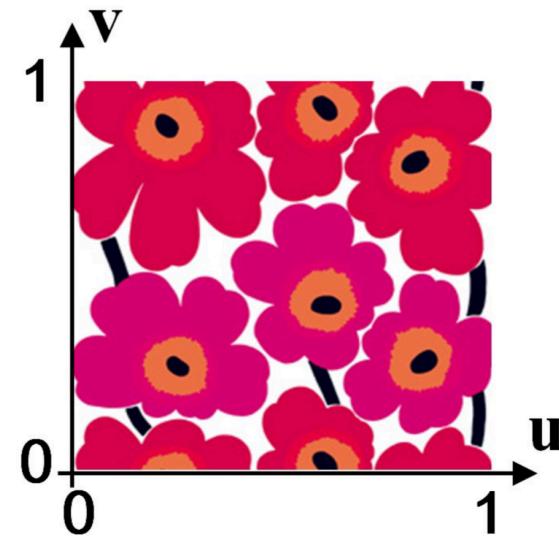
- **Grundsatz:** Die Herkunft einer Textur ist zweitrangig; wichtig ist nur, dass sie an allen benötigten Stellen definiert und abrufbar ist.
- **Standardverfahren:** Meist wird eine Textur in einem Vorverarbeitungsschritt als **Pixel-Array** (also ein Raster von Farbwerten) erstellt und später im Rendering-Prozess darauf zugegriffen.
- **Mögliche Quellen für Texturen:**
 - **Fotografien:** Man kann Fotos von realen Oberflächen verwenden.
 - **Scans:** Gescanntes Material.
 - **Programmgenerierte Texturen:** Texturen, die durch ein Programm erzeugt werden, bis hin zu Zufallswerten (Noise).
 - **Datenbanken:** Für häufig verwendete Texturen (z.B. Holzmaserung, Gras, Sand, Marmor, Kopfsteinpflaster, Stoffstrukturen) können Datenbanken angelegt werden.
- **Prozedurale Texturierung:**
 - **Definition:** Wenn Texturen aus einer **mathematischen Funktion** gewonnen werden, spricht man von "Procedural Texturing".
 - **Vorteile von Prozeduralen Texturen:**
 - **Kein Speicherplatz für Bilder:** Die Textur wird zur Laufzeit berechnet, nicht gespeichert.
 - **Unendliche Detailtiefe:** Können beliebig hoch aufgelöst werden, ohne an Qualität zu verlieren oder Pixelartefakte zu zeigen (im Gegensatz zu Bitmap-Texturen).
 - **Parameterisierbarkeit:** Oft lassen sich Parameter der Funktion ändern, um Variationen der Textur zu erzeugen.



Textur Objekt Transformation

[EVC_Skriptum_CG, p.47](#)

- **Ausgangssituation:**
 - Die Textur liegt normalerweise in einem **2D-Koordinatensystem** vor, dessen Achsen mit (u, v) bezeichnet werden. Diese (u, v) -Koordinaten sind die Texturkoordinaten.
 - Die Oberfläche des 3D-Objekts, auf die die Textur aufgebracht werden soll, hat ebenfalls eine **parametrische Darstellung**, die mit (u^*, v^*) bezeichnet wird. Diese (u^*, v^*) -Koordinaten sind die Oberflächenparameter des Objekts.

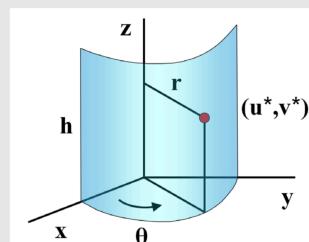


- **Ziel der Transformation:** Für jeden Punkt auf der 3D-Objektoberfläche (identifiziert durch seine (u^*, v^*) -Parameter) soll die zugehörige Farbe aus der Textur ermittelt werden. Das bedeutet, wir brauchen eine Abbildung von den Objekt-Oberflächenparametern zurück zu den Texturkoordinaten, oder umgekehrt, eine Abbildung von den Texturkoordinaten zu den Objekt-Oberflächenparametern.
- **Biliniere Funktion für die Textur-Objekt-Transformation:**
 - Eine häufig verwendete Methode, um eine Textur auf eine Oberfläche aufzubringen, ist eine bilinare Funktion. Diese Funktion bildet die Texturkoordinaten (u, v) auf die Oberflächenparameter (u^*, v^*) ab:
 - $u^* = u^*(u, v) = a_u u + b_u v + c_u$
 - $v^* = v^*(u, v) = a_v u + b_v v + c_v$
 - **Erklärung:** Diese linearen Gleichungen (bilinear, wenn man die unabhängigen Variablen u, v und die Konstanten $a_u, b_u, c_u, a_v, b_v, c_v$ betrachtet) definieren, wie ein Punkt in der 2D-Textur (u, v) auf einen Punkt auf der 3D-Oberfläche (u^*, v^*) abgebildet wird. Die Koeffizienten (a, b, c) steuern dabei Skalierung, Rotation, Scherung und Translation der Textur auf der Oberfläche.
 - **Praktische Bedeutung:** Man kann für jeden Punkt der Objektoberfläche die zugehörige Farbe aus der Textur bestimmen.
- **Bezeichnung:** Diese Funktion wird als **Textur-Objekt-Transformation** bezeichnet und mit M_T denotiert.

Beispiel:

Eine Textur $t(u, v)$, $0 \leq u, v \leq 1$, soll auf einen Viertel-Zylinder mit Höhe h aufgetragen werden, dessen Mantel in z -Richtung mit v^* parametrisiert ist und entlang der Krümmung mit $u^* (= \theta)$. Um nun für ein Texturpixel $t(u, v)$ [auch Texel genannt] zu berechnen, an welche Stelle des Zylinders es zu liegen kommt, muss man die Abbildung M_T definieren, das könnte etwa sein:

$$u^* = u \cdot \pi/2, v^* = v \cdot h \quad (\text{so passt die Textur genau auf das Zylinderviertel}).$$



Viewing und Projektionstransformation

- Die Abbildung eines 3D-Modells auf eine Bildebene ist grundsätzlich eine einfache *Projektion M_{VP}* .
- Beim *Rasterscannen* von Flächen (dem Prozess des Füllens von Pixeln), um sicherzustellen, dass jedes Pixel *genau einmal* gefärbt wird (also keine Übermalungen oder Löcher), arbeitet man in umgekehrter Richtung:
 - Man bestimmt für jedes Pixel (x, y) auf der Bildebene, welcher Oberflächenpunkt dort gezeichnet wird.
 - Dazu werden die (u_*, v_*) -Koordinaten (Texturkoordinaten) der Fläche und daraus das *Texel* (Textur-Pixel) für dieses Pixel bestimmt.
 - Dafür werden die inversen Operatoren M_{VP}^{-1} und M_T^{-1} benötigt.

Beispiel (Fortsetzung):

Für eine beliebige *Projektion* reicht es, wenn wir von jedem Punkt die (x, y, z) -Koordinaten kennen. Im Falle des obigen Zylinders ergibt sich: $x = r \cdot \cos u*$, $y = r \cdot \sin u*$, $z = v*$.

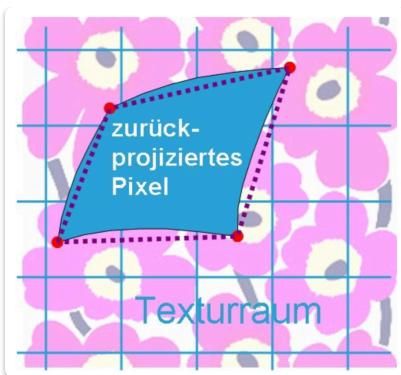
Nun wird das Problem von hinten angegangen:

- Für einen Bildpunkt P bestimmt man zuerst die Position (x, y, z) am Zylinder, die dort dargestellt wird (z.B. durch Ray-Casting).
- Für diesen Punkt muss man die Parameter der Oberfläche finden: $u^* = \cos^{-1}(x/r)$, $v^* = z$. Bis hier ist das also die inverse Transformation M_{VP}^{-1} .
- Jetzt muss für das Parameterpaar (u^*, v^*) noch die Textur gefunden werden, indem M_T invertiert wird: $u = 2u^*/\pi$, $v = v^*/h$ (das ist M_T^{-1})

Anti-Aliasing für Texturen

[EVC_Skriptum_CG, p.48](#)

- **Problem:** Texturen sind besonders anfällig für *Aliasing-Effekte*. Dies tritt verstärkt auf, wenn Texturmuster vergrößert (Verpixelung) oder verkleinert (Moiré-Effekte, Flackern) werden.
 - **Aliasing:** Unerwünschte Artefakte (wie Treppeneffekte, Flackern, Moiré-Muster), die entstehen, wenn ein kontinuierliches Signal (die Textur) mit einer zu geringen Abtastrate (den Pixeln) diskretisiert wird.
- **Korrekte, aber langsame Lösung:**
 - Man müsste den *Textur-Durchschnittswert* der Fläche berechnen, die von einer Rückprojektion des zu füllenden Pixels auf die Textur erzeugt wird.
 - **Rückprojektion:** Ein Pixel auf dem Bildschirm entspricht nicht einem einzelnen Punkt, sondern einer Fläche in der Textur. Man muss diese Fläche in der Textur bestimmen.
 - **Näherung:** Oft reicht es näherungsweise aus, das Viereck zu verwenden, das durch die gerade Verbindung der rückprojizierten Eckpunkte des Pixels entsteht.
 - **Nachteil:** Diese Methode ist sehr langsam.



- Optimierungen für Anti-Aliasing bei Texturen:

1. Mip-Mapping:

- **Prinzip:** Die Textur wird in **verschiedenen Größen (Auflösungen)** vorab berechnet und gespeichert (ein sogenannter "Mip-Map-Pyramide").
- **Anwendung:** Je nachdem, wie stark ein Objekt im Bild verkleinert erscheint, wird die passende Texturgröße aus der Mip-Map-Pyramide ausgewählt.
- **Qualitätsverbesserung:** Zwischen den verschiedenen Größenstufen kann dann linear interpoliert werden, um fließende Übergänge und eine bessere Qualität zu erzielen.
- **Vorteil:** Reduziert Aliasing bei Verkleinerung der Textur und verbessert die Cache-Kohärenz.

2. Summed-Area-Table-Methode (SAT):

- **Prinzip:** Man erstellt eine sogenannte **Summen-Textur (Summed-Area-Table)**. In jedem Punkt dieser Tabelle ist die Summe aller Textelwerte (Textur-Pixel) von einem Referenzpunkt bis zu diesem Punkt gespeichert.
- **Anwendung:** Durch einfache Differenzenbildung in dieser Summen-Textur kann man leicht den Durchschnittswert **beliebiger rechteckiger Bereiche** in der Originaltextur ermitteln.
- **Vorteil:** Ermöglicht die schnelle Berechnung von Durchschnittswerten für rechteckige Texturbereiche, was für die Filterung von Texturen beim Anti-Aliasing nützlich ist.
- **Einschränkung:** Bei nicht-rechteckigen rückprojizierten Pixeln kann die Summed-Area-Table nur eine Approximation liefern.

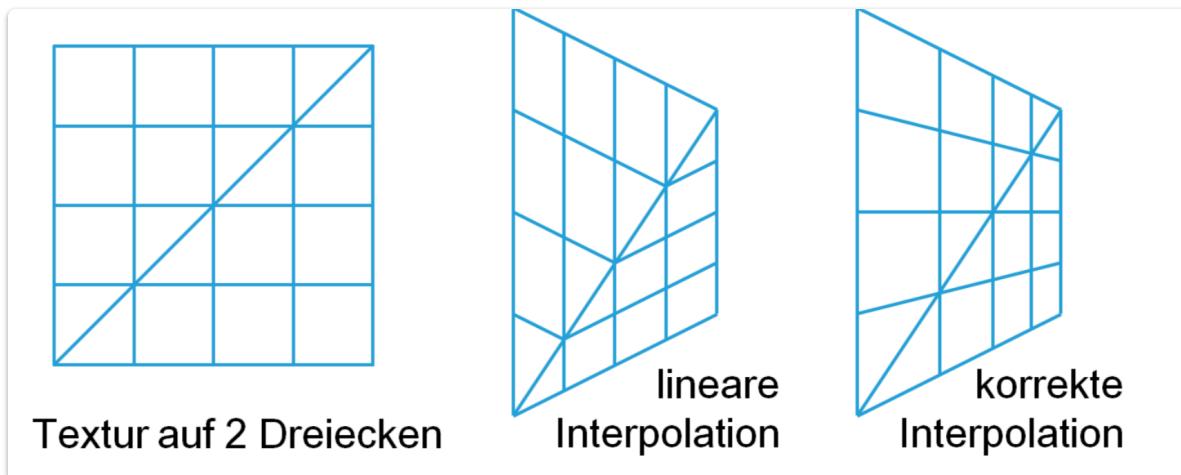
Texturen auf perspektivisch verzerrten Dreiecken

[EVC_Skriptum_CG, p.48](#)

Die Verwendung von **baryzentrischen Koordinaten** ermöglicht eine **lineare Interpolation der Eckpunktwerte** über die Dreiecksfläche.

Bei der **perspektivischen Transformation** geht diese Linearität der Oberflächenparameter jedoch verloren. Daher muss die Interpolation vor der Homogenisierung erfolgen, da sie dort nicht linear ist.

Anstatt die Farbe für einen Punkt $p(\alpha, \beta, \gamma)$ mit $\text{color}(x, y) = \alpha \cdot t_0 + \beta \cdot t_1 + \gamma \cdot t_2$ zu berechnen, werden die **Texturparameter u und v** mithilfe von **baryzentrischen Koordinaten ($\alpha_w, \beta_w, \gamma_w$) vor der Perspektive** ermittelt. Diese werden dann zur Berechnung des Texturwertes am Punkt $p(\alpha, \beta, \gamma) = p(x, y)$ verwendet.



- **Lineare Interpolation (nicht korrekt für perspektivisch verzerrte Dreiecke):** Wenn man die Texturparameter u und v direkt nach der perspektivischen Transformation linear interpoliert, entstehen Verzerrungen, wie im mittleren Bild gezeigt. Die Textur "verläuft" nicht gleichmäßig über die Fläche.
- **Korrekte Interpolation:** Um dies zu vermeiden, werden die Texturparameter u und v (oder genauer, die baryzentrischen Koordinaten $\alpha_w, \beta_w, \gamma_w$, die zu u, v führen) **vor** der perspektivischen Transformation berechnet. Dadurch bleibt die Linearität erhalten und die Textur wird korrekt auf das perspektivisch verzerrte Dreieck abgebildet.

Formeln für die korrekte Texturinterpolation:

Gegeben sind die baryzentrischen Koordinaten α, β, γ des Punkts $p(x, y)$ im perspektivisch transformierten Raum. Um die korrekten baryzentrischen Koordinaten $\alpha_w, \beta_w, \gamma_w$ im "World-Space" (vor der Projektion) zu finden, benötigen wir die homogenen Koordinaten und die w -Komponente (h_0, h_1, h_2) .

Die Formeln lauten:

$$d = h_1 h_2 + h_2 \beta (h_0 - h_1) + h_1 \gamma (h_0 - h_2)$$

$$\beta_w = h_0 h_2 \beta / d$$

$$\gamma_w = h_0 h_1 \gamma / d$$

$$\alpha_w = 1 - \beta_w - \gamma_w$$

- **Erklärung der Variablen:**

- h_0, h_1, h_2 : Dies sind wahrscheinlich die w -Komponenten (homogene Koordinaten) der Eckpunkte des Dreiecks **vor** der Division durch w (d.h., nach der Multiplikation mit der Projektionsmatrix, aber bevor die einzelnen Koordinaten durch w geteilt

werden). In der Praxis werden diese w -Werte oft als w_0, w_1, w_2 der Eckpunkte v_0, v_1, v_2 bezeichnet.

- d : Ein Hilfswert, der für die Berechnung von β_w und γ_w benötigt wird.
- $\alpha_w, \beta_w, \gamma_w$: Die korrigierten baryzentrischen Koordinaten im "World-Space".

Sobald wir die korrigierten baryzentrischen Koordinaten $\alpha_w, \beta_w, \gamma_w$ haben, können wir die Texturparameter u und v interpolieren:

$$u = \alpha_w u_0 + \beta_w u_1 + \gamma_w u_2$$

$$v = \alpha_w v_0 + \beta_w v_1 + \gamma_w v_2$$

- **Erklärung der Variablen:**

- $u_0, v_0, u_1, v_1, u_2, v_2$: Die Texturkoordinaten der drei Eckpunkte des Dreiecks.

Zuletzt wird die Farbe des Punktes mit den berechneten Texturparametern u und v bestimmt, indem der Texturwert aus einer Texturfunktion $t(u, v)$ abgefragt wird:

$$\text{color}(x, y) = t(u, v)$$

Solid Texturing

[EVC_Skriptum_CG, p.49](#)

Solid Texturing ist eine Methode, bei der eine Textur nicht als 2D-Bild, sondern als **3D-Volumen** definiert ist.

- **Wie es funktioniert:**
 - In einem **3-dimensionalen Parameterraum** wird für jeden Raumpunkt eine Farbe oder ein Texturwert definiert.
 - Wenn sich eine Oberfläche an einem bestimmten Raumpunkt befindet, wird die dort definierte Farbe/der Texturwert abgerufen und auf die Oberfläche angewendet.
- **Darstellung der Textur:**
 - Die 3D-Textur kann entweder als **mathematische Funktion** gegeben sein (z.B. eine Perlin Noise Funktion für Wolken oder Marmor).
 - Oder sie kann durch **Volumendaten** repräsentiert werden (ähnlich wie ein 3D-Gitter, bei dem jeder Voxel einen Farbwert speichert).
- **Wichtige Voraussetzung:** Man muss in der Lage sein, die Texturwerte für **jeden Raumpunkt** abzufragen.



Vorteile von Solid Texturing:

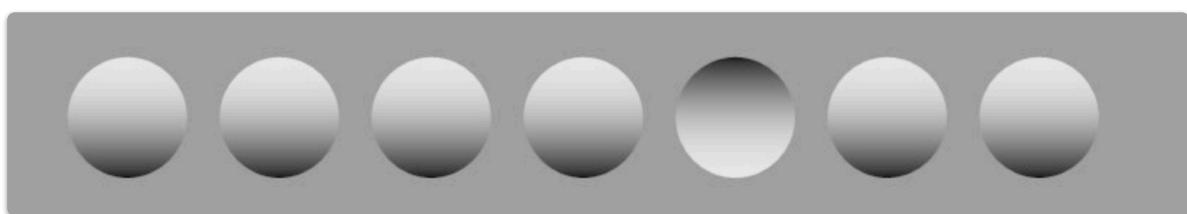
1. **Kohärentes Muster über Kanten hinweg:** Der größte Vorteil ist, dass das Texturmuster **nahtlos über alle Kanten** und zwischen verschiedenen Polygonen eines Objekts verläuft.
 - **Keine Zusammenfügbungsprobleme:** Bei traditionellem 2D-UV-Mapping kann es zu sichtbaren Nähten oder Verzerrungen an den Grenzen von UV-Segmenten kommen. Solid Texturing eliminiert dieses Problem, da die Textur "durch" das Objekt geht, als wäre es aus dem Texturvolumen geschnitten.
2. **Einfachere Abbildung der Textur auf das Objekt:** Die Zuordnung der Textur zu den Objektkoordinaten ist wesentlich einfacher zu handhaben, da sie direkt auf den 3D-Positionen des Objekts basiert und nicht erst komplexe 2D-UV-Koordinaten berechnet werden müssen.

Bump Mapping

[EVC_Skriptum_CG, p.49](#)

Viele Oberflächen in der realen Welt besitzen eine **geometrische Detailstruktur** (z.B. die Rinde eines Baumes, eine Münzoberfläche, Stoffgewebe, Putz an einer Wand). Solche feinen Details auf der Oberfläche als tatsächliche Geometrie (d.h. durch zusätzliche Polygone) zu modellieren, ist extrem aufwendig und erzeugt riesige Datenmengen.

Bump Mapping ist eine Technik, um diesen Aufwand signifikant zu reduzieren.

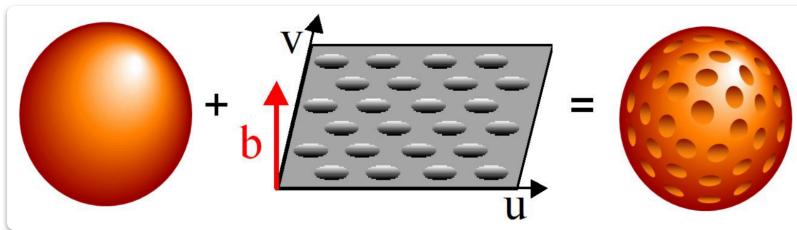


Betrachtet man die graue Leiste, scheint sie sechs Ausbuchtungen und eine Einbuchtung zu haben. Fühlt man die Oberfläche jedoch an, stellt man fest, dass sie vollkommen eben ist.

- **Warum sehen wir die Unebenheiten?**

- Der Eindruck von Unebenheiten entsteht allein durch die **Schattierung**. Unser Gehirn interpretiert die Helligkeits- und Schattenmuster als dreidimensionalen Raum.

Grundidee des Bump Mappings:



Die Grundidee des Bump Mappings besteht darin, den **Eindruck von Oberflächenunebenheiten (Bumps)** zu erwecken, ohne die tatsächliche Geometrie der Oberfläche zu verändern.

- Wie es funktioniert:**
 - Die geometrische Oberfläche bleibt unverändert (z.B. eine glatte Kugel oder Ebene).
 - Stattdessen wird der **Normalvektor** der Oberfläche (der für die Lichtberechnung verwendet wird) entsprechend der gewünschten Unebenheit **modifiziert**.
 - Diese Modifikation des Normalvektors beeinflusst, wie das Licht von der Oberfläche reflektiert wird, was wiederum die Schattierung verändert.
 - Dadurch entspricht die **Schattierung den "Bumps"**, obwohl geometrisch nichts an der Oberfläche verändert wurde.
- Vorteil:** Man kann mit sehr geringem Aufwand (nur durch die Anpassung der Normalen in der Fragment-Shader-Phase) den visuellen Eindruck von komplexen Details erzeugen, ohne die Anzahl der Polygone erhöhen zu müssen.

Bump-Mapping-Algorithmus

EVC_Skriptum_CG, p.49

Sei die Bump-Textur in Form eines Arrays von Höhenwerten $b(u, v)$ gegeben, das heißt also, dass die Position der Stelle $p(u, v)$ der Oberfläche, die durch das Parameterpaar (u, v) erzeugt wird, um $b(u, v)$ in Richtung des Normalvektors n an dieser Stelle verschoben erscheinen soll. n erhält man indem man das Kreuzprodukt zweier Tangentenvektoren auf die Länge 1 normiert:

$$n^* = p_u \times p_v, \quad n = n^* / |n^*|$$

Der verschobene Punkt $p'(u, v)$ ergibt sich dann zu: $p'(u, v) = p(u, v) + b(u, v) \cdot n$. Wir aber brauchen n' , also die Normale auf den verschobenen Punkt:

$$n' = p'_u \times p'_v$$

Nun gilt:

$$p'_u = \partial(p + b_n) / \partial u = p_u + b_u n + b n_u$$

und weil b sehr klein ist: $p'_u \approx p_u + b_u n$ analog gilt natürlich $p'_v \approx p_v + b v n$, sodass sich n' ergibt:

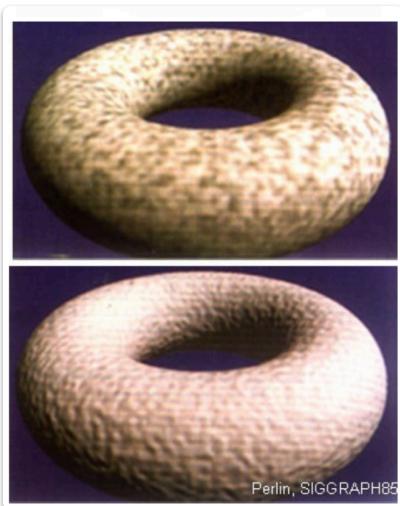
$$n' = p'_u \times p'_v = p_u \times p_v + b_v(p_u \times n) + b_u(n \times p_v) + b_u b_v(n \times n)$$

und aus $n \times n = 0$ folgt schließlich: $n' = n + b_v(p_u \times n) + b_u(n \times p_v)$.

Für die Berechnung des modifizierten Normalvektors benötigt man nicht den Höhenwert $b(u, v)$ direkt, sondern die **Ableitungen** von $b(u, v)$ nach u und v (also $\frac{\partial b}{\partial u}$ und $\frac{\partial b}{\partial v}$).

- **Praktische Anwendung:**

- In der Praxis sind die Parametrisierung der Oberfläche (UV-Koordinaten) und die der Bump-Textur häufig identisch.
- Das ermöglicht es, diese Ableitungen **vorzuberechnen** und direkt in der Bump-Map zu speichern, anstatt die eigentlichen Höhenwerte $b(u, v)$ zu speichern. Eine solche Map, die Ableitungen oder direkt die Normalen speichert, nennt man oft **Normal Map**.



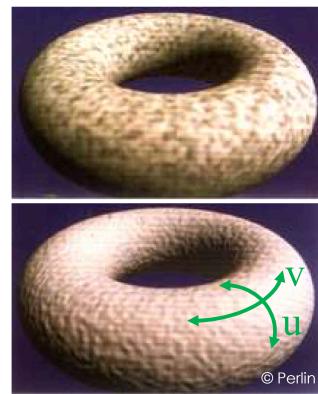
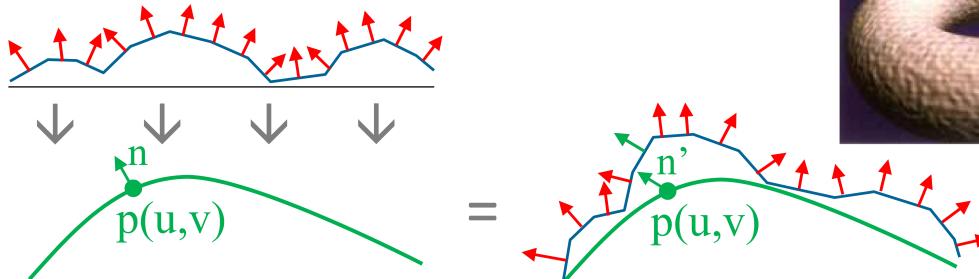
Beachte den Unterschied in der Donut-Abbildung:

- **Texture-Map (oben):** Zeigt die reinen Farb- oder Helligkeitsinformationen der Textur.
- **Bump-Map (unten):** Ist eine Darstellung, die die lokalen Unebenheiten kodiert.

surface roughness is simulated

perturbation function varies surface normal *locally*

bump map $b(u,v)$ derivative $b'(u,v)$



Der räumliche Eindruck der Oberfläche entsteht erst, wenn die **Schattierung eine Richtungsabhängigkeit bekommt**, d.h., wenn das Licht aus verschiedenen Richtungen unterschiedlich reflektiert wird, basierend auf den modifizierten Normalen.

Einschränkungen und Nachteile von Bump Mapping

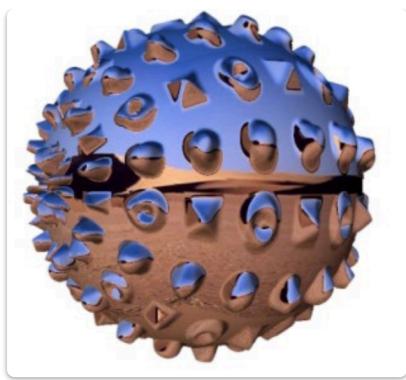
[EVC_Skriptum_CG](#), p.50

Bump Mapping ist ein **visueller Trick**, der die Schattierung verändert, aber die tatsächliche Geometrie nicht korrigiert. Dies führt zu **sichtbaren Fehlern**, die bei ausgeprägten "Bumps" deutlicher werden:

1. **Verzerrung bei flachen Winkeln:** Bei flachen Blickwinkeln erscheint die simulierte Struktur stark verzerrt und unrealistisch.
2. **Glatte Silhouette:** Der **Umriss des Objekts bleibt glatt**, da die Geometrie unverändert ist, was bei Objekten mit echten Unebenheiten falsch aussieht.
3. **Glatte Schattenränder:** Bedingt durch die glatte Silhouette wirft das Objekt **Schatten mit glatten Rändern**, statt unregelmäßigen.
4. **Keine gegenseitigen Schatten der Bumps:** Die simulierten Unebenheiten **werfen keine Schatten aufeinander** oder auf die Oberfläche selbst, da sie keine echte Geometrie besitzen.
5. **Falsche Beleuchtung auf lichtabgewandter Seite:** Modifizierte Normalen können dazu führen, dass Oberflächenbereiche, die geometrisch im Schatten liegen sollten, **fälschlicherweise beleuchtet werden**.

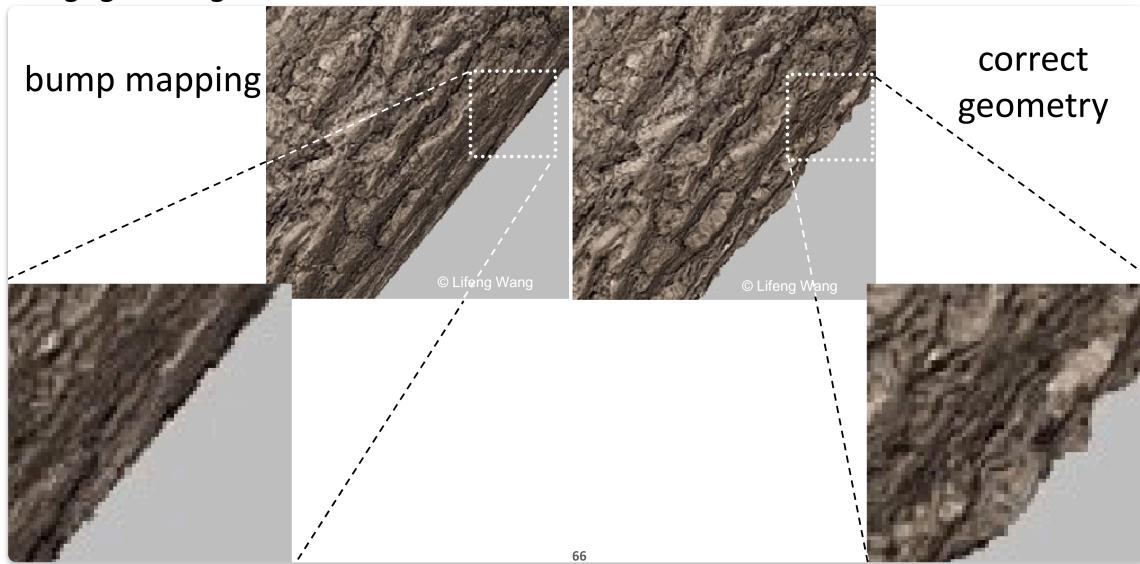
Displacement Mapping

Während Bump Mapping eine Vielzahl von Fehlern aufweist (wie in den vorherigen Notizen beschrieben), gibt es für jeden dieser Fehler mehr oder weniger aufwendige Hilfslösungen. Die **korrekteste Methode** zur Darstellung von Oberflächenunebenheiten ist jedoch, die **Oberfläche tatsächlich um die "Bump-Höhe" zu verändern**. Diese Methode wird als **Displacement Mapping** bezeichnet.



Funktionsweise und Vorteile:

- **Tatsächliche Geometrieverziehung:** Bei Displacement Mapping werden die Oberflächenpunkte tatsächlich verschoben. Das bedeutet, dass die geometrischen Koordinaten der Scheitelpunkte (Vertices) des Modells verändert werden, um die Unebenheiten zu repräsentieren.
- **Korrekte Silhouette:** Da die Geometrie physisch verändert wird, entsteht natürlich auch eine korrekte Silhouette des Objekts. Die Ränder des Objekts zeigen nun die tatsächlichen Unebenheiten und nicht mehr die glatte Form der ursprünglichen Geometrie.
- **Korrekte Schatten:** Die versetzte Geometrie wirft auch korrekte Schatten, einschließlich der gegenseitigen Schatten der Unebenheiten aufeinander.



66

Implementierung und Hardware-Unterstützung:

EVC_Skriptum_CG, p.50

- **Aufwand:** Displacement Mapping ist viel aufwendiger zu implementieren und rechenintensiver als Bump Mapping, da es eine tatsächliche Veränderung der Geometrie erfordert (oft durch Hinzufügen vieler neuer Polygone).
- **Hardware-Unterstützung:** Moderne Grafikkarten unterstützen Displacement Mapping jedoch. Dies geschieht typischerweise mittels einer zusätzlichen programmierbaren Hardware-Stufe in der Rendering-Pipeline, die oft als "Tessellation-Stage" bezeichnet wird.

- **Tessellation-Stage:** Diese Stufe befindet sich **zwischen dem Vertex- und dem Pixel-Stage**. Ihre Aufgabe ist es, die Dreiecke direkt auf der Hardware in **viele kleine Dreiecke zu unterteilen** (Tessellation). Jedes dieser neu erzeugten kleinen Dreiecke kann dann entsprechend einer **Displacement Map** verschoben werden, um die feinen Details zu erzeugen.

Zusammenfassend: Displacement Mapping bietet eine physikalisch korrektere Darstellung von Oberflächenstrukturen als Bump Mapping, indem es die Geometrie tatsächlich modifiziert. Dies ist zwar komplexer, wird aber durch moderne GPU-Hardware effizient unterstützt.

Kombination mehrerer Mappings

EVC_Skriptum_CG, p.50

- **Multitexturing** ist eine mächtige Methode, um *mehrere Mappings zu kombinieren*.
- Beispiele für kombinierbare Texturen sind:
 - Grundmuster (z.B. Holzmaserung)
 - Beleuchtung (z.B. Lichtkegel)
 - Verschmutzung
 - Unebenheiten
 - Fotos plus Annotationen (zusätzliche Informationen oder Markierungen auf einem Bild)

