Texturen und andere Mappings

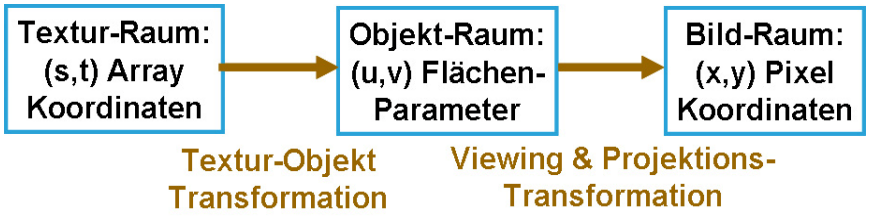
Eine wesentliche optische Eigenschaft natürlicher Oberflächen ist, dass sie diverse Unregelmäßigkeiten aufweisen. Das kann entweder durch die Umgebung verursacht sein, oder durch eine variable Färbung der Oberfläche, oder durch die Schattierung, die von Oberflächenunebenheiten her rührt. Für diese drei Ursachen lassen sich die Effekte mit den drei Methoden *Environment Mapping*, *Texture Mapping* und *Bump Mapping* simulieren. Mapping bedeutet dabei *Abbilden*.

Environment Mapping

Environment bezeichnet die Umgebung, die Umwelt eines Objektes. Je nach Oberflächeneigenschaften des Objektes wird sich die Umgebung verschieden auf das Erscheinen des Objektes auswirken. Für perfekt spiegelnde Oberflächen können wir mit Ray-Tracing das exakte Spielgelbild der Umgebung erzeugen. Für nicht perfekt diffuse Oberflächen können wir mit dem Phong-Modell Glanzeffekte in der Umgebung der Spiegelung von Lichtquellen annähern. Eine mehr oder weniger spiegelnde Oberfläche reflektiert aber ihre ganze Umgebung mehr oder weniger scharf, wobei die Genauigkeit sehr reduziert ist. Um nun effizient Objekte in einer komplexen Umgebung zu rendern ohne diese ganze Umgebung vollständig modellieren zu müssen und ohne Anspruch auf Exaktheit, verwendet man Environment Mapping. Dazu wird die Umgebung in einem Vorverarbeitungsschritt von einem zentralen Punkt aus (z.B. dem Mittelpunkt des Objektes oder der darzustellenden Szene) als Bild produziert. Ob das nun ein berechnetes oder ein photographiertes Bild ist, spielt keine Rolle. Jedenfalls geht man davon aus, dass die Umgebung (z.B. Kugel oder Würfel) groß ist im Verhältnis zu den Objekten, die man darstellen will. Nun wird bei der Darstellung eines Objektes für jeden Oberflächenpunkt näherungsweise angenommen, dass er im Zentrum dieser Umgebung liegt. Dadurch kann man allein aus der Richtung des Reflexionsstrahles schnell bestimmen, welcher Umgebungspunkt getroffen wird (z.B. mit Polarkoordinaten), und erspart sich aufwändiges Ray-Tracing.

Texture Mapping

Viele Oberflächen sind nicht einfärbig, sondern haben ein Muster, z.B. Holzmaserung, Bilder an der Wand, Schrift auf Papier, Verschmutzung, Kleider, Marmor. Auch bei grober Modellierung lassen sich Details als Muster interpretieren, z.B. Fenster auf einer Hauswand, Wolken am Himmel, Gesichter, Knöpfe und Reißverschlüsse, Pflastersteine. Solche Muster bezeichnet man als Textur. Das Aufbringen von Texturen wird Texture Mapping genannt.



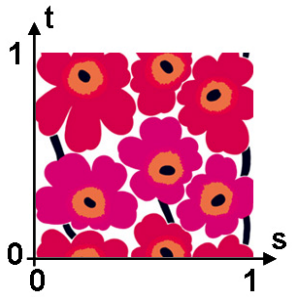
Nach der Erzeugung der Texturen erfolgt Texture Mapping in zwei Schritten. Zuerst muss definiert werden, welche Textur auf welche Oberfläche der Objekte wie orientiert, skaliert etc. aufzubringen ist. Dies ist eigentlich ein Teil der Modellierung, wo die Oberflächen ein Aussehen erhalten. Und im zweiten Schritt muss dann die Textur korrekt auf das Abbild der Objekte gerendert werden, also in das Bild transformiert werden.

Erzeugung einer Textur

Grundsätzlich ist es egal, woher eine Textur kommt, sie muss lediglich an allen Stellen definiert und abrufbar sein. Meist wird eine Textur in einem Vorverarbeitungsschritt als Pixel-Array hergestellt und dann nur noch darauf

zugegriffen. Man kann also eine Photographie ebenso verwenden wie einen Scan, aber auch eine durch ein Programm erzeugte Textur bis hin zu Zufallswerten. Für häufig verwendete Texturen wie Holzmaserung, Gras, Sand, Marmor, Kopfsteinpflaster oder Stoffstrukturen kann man sich auch eine Datenbank anlegen. Verwendet man Texturen, die aus einer mathematischen Funktion gewonnen werden, so nennt man das auch „Procedural Texturing".

Textur-Objekt-Transformation



Normalerweise wird die Textur in einem 2D-Koordinatensystem vorliegen, den wir mit (s,t) ansprechen wollen. Weiters wollen wir annehmen, dass eine Oberfläche, auf die die Textur aufgebracht werden soll, ebenfalls eine parametrische Darstellung hat, die wir mit (u,v) bezeichnen. Eine bilineare Funktion zum Aufbringen der Textur sieht dann so aus:

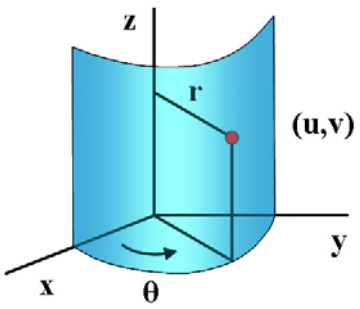
$$u = u(s,t) = a_u s + b_u t + c_u \,, \qquad v = v(s,t) = a_v s + b_v t + c_v$$

d.h. man kann für jeden Punkt der Objektfläche die zugehörige Farbe ermitteln. Diese Funktion heißt Textur-Objekt-Transformation und wird mit $\mathbf{M_T}$ denotiert.

Beispiel:

Eine Textur T(s,t), $0 \le s,t \le 1$, 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 (= θ). Um nun für ein Texturpixel T(s,t) [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:

$u = s \cdot \pi/2$, $v = t \cdot h$ (so passt die Textur genau auf das Zylinderviertel).



Viewing und Projektionstransformation

An sich ist die Abbildung des 3D-Modells auf eine Bildebene eine einfache Projektion $\mathbf{M_{VP}}$. Um beim Rasterscannen der Flächen jedes Pixel genau einmal zu färben (also keine Übermalungen zu machen und keine Löcher zu lassen), arbeitet man nun in umgekehrter Richtung. Man bestimmt für jedes Pixel (x,y) welcher Oberflächenpunkt dort gezeichnet wird (also die (u,v)-Koordinaten der Fläche) und dann daraus, welches Texel für dieses Pixel Gültigkeit hat. Dazu braucht man die inversen Operatoren $\mathbf{M_{VP}^{-1}}$ und $\mathbf{M_T^{-1}}$,

Beispiel (Fortsetzung):

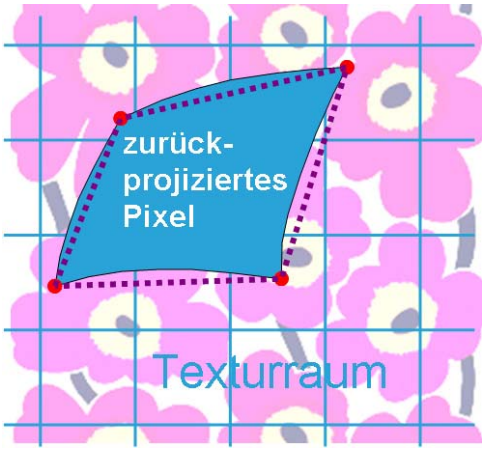
Für eine beliebige Projektion reicht es, wenn wir von jedem Punkt die (x,y,z)-Koordinaten kennen. Im Falle des obigen Zylinders ergibt sich: $x = r \cdot \cos u$, $y = r \cdot \sin u$, $z = v$.

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:
  $s = 2u/\pi$, $t = v/h$ (das ist M_T^{-1})

Anti-Aliasing für Texturen

Texturen sind für Aliasing-Effekte besonders anfällig, insbesondere wenn die Muster vergrößert oder verkleinert werden. Korrekt müsste man den Textur-Durchschnittswert der Fläche berechnen, die von einer Rückprojektion des zu füllenden Pixels auf die Textur erzeugt wird. Näherungsweise reicht auch das Viereck, das durch gerade Verbindung der rückprojizierten Eckpunkte entsteht. Da dies sehr langsam wäre, verwendet man häufig eine von zwei Optimierungen: *Mip-Mapping*, bei dem die Textur in verschiedenen Größen vorberechnet und dann linear interpoliert wird, und die *Summed-Area-Table*-Methode, wo man aus einer Summentextur durch Differenzenbildung leicht die Durchschnittswerte rechteckiger Bereiche ermittelt.



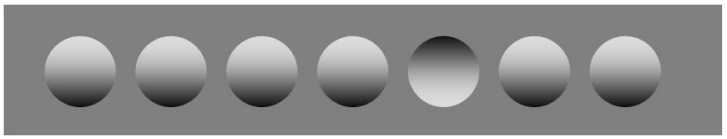
Solid Texturing

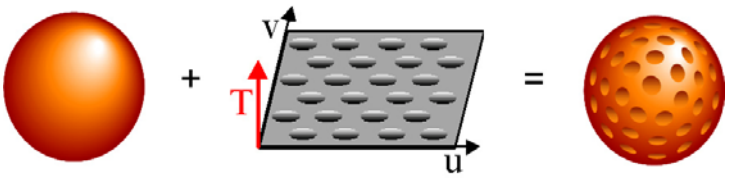
Eine Textur kann außer in 2-dimensionaler Form auch als 3D-Volumen gegeben sein. Das heißt, es wird in einem 3-dimensionalen Parameterraum für jeden Raumpunkt eine Farbe definiert, die dann von einer Oberfläche, die sich an diesem Raumpunkt befindet, abgerufen wird. Bei dieser Methode kann die Textur entweder als mathematische Funktion gegeben sein oder durch Volumendaten, wichtig ist nur, dass man die Werte für jeden Raumpunkt abfragen kann. Der große Vorteil von Solid Texturing besteht darin, dass die Muster über alle Kanten kohärent weiterlaufen, dass es also keine Zusammenfügungsprobleme zwischen irgendwelchen Polygonen geben kann. Weiters ist natürlich die Abbildung der Textur auf das Objekt wesentlich einfacher zu handhaben.

Bump Mapping und Displacement Mapping

Viele Oberflächen haben eine geometrische Detailstruktur: Rinde, Münzen, Verputz, Leder, Stoff, Gemüse, Planeten, Schotterwege, Lasagne, Fliesen, Schokoladetafel, Regenwurm usw. Solche Objekte komplett zu modellieren ist sehr mühsam und erzeugt riesige Datenmengen. Mit Bump-Mapping kann man diesen Aufwand signifikant reduzieren.

Wenn man die graue Leiste rechts ansieht, so hat man den Eindruck, dass sie sechs Ausbuchtungen und eine Einbuchtung hat. Wenn man sie aber angreift, ist sie ganz eben! Warum sehen wir die Unebenheiten? Weil die Schattierung alleine ausreicht um einen räumlichen Eindruck zu erwecken! Diesen Trick kann man verwenden, um mit wenig Aufwand den Eindruck von Oberflächenunebenheiten (Bumps) zu erwecken. Die Grundidee ist es, die Oberfläche unverändert zu lassen, aber den Normalvektor entsprechend der Unebenheiten zu verändern. Dadurch entspricht die Schattierung den Bumps, aber geometrisch braucht man nichts verändern.



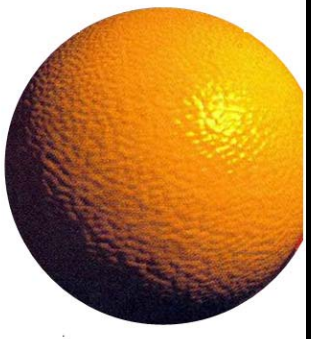


Bump-Mapping-Algorithmus

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:

$$N = P_u \times P_v, \qquad n = N / |N|$$

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:

$$P_u' = \partial(P + bn)/\partial u = P_u + b_u n + b n_u \quad \text{und weil b sehr klein ist:} \quad P_u' \approx P_u + b_u n$$

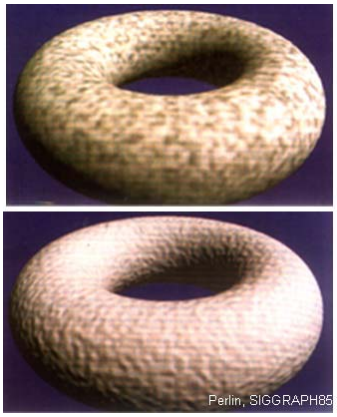
analog gilt natürlich $P_v' \approx P_v + b_v n$, sodass sich N' ergibt:

$$N' = P_u' \times P_v' = P_u \times P_v + b_v(P_u \times n) + b_u(n \times P_v) + b_u b_v(n \times n)$$

und aus $n \times n = 0$ folgt schließlich :

$$\mathbf{N' = N + b_v(P_u \times n) + b_u(n \times P_v)}$$

Man braucht also in Wahrheit nicht b(u,v) sondern die Ableitungen nach u und v. Da in der Praxis die Parametrisierung der Oberfläche und der Bump-Textur häufig gleich sind, kann man diese Ableitungen leicht vorberechnen und statt der b(u,v) abspeichern.



Man beachte in der Donut-Abbildung rechts den Unterschied zwischen einer Texture-Map (oben) und einer Bump-Map (unten). Der räumliche Eindruck der Oberfläche entsteht erst, wenn die Schattierung eine Richtungsabhängigkeit bekommt.

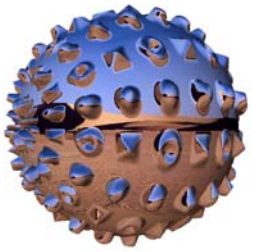
Natürlich ist Bump-Mapping lediglich ein großer Schwindel, bei dem die Schattierung verändert wird ohne die Geometrie zu korrigieren. Dementsprechend verbleiben auch sichtbare Fehler, die umso deutlicher werden, je höher die Bumps sind:

1. bei flachen Winkeln wird die Struktur stark verzerrt
2. die Silhouette bleibt so glatt wie die ursprüngliche Geometrie ist (also glatt = falsch)
3. daher wirft das Objekt auch Schatten mit glatten (= falschen) Rändern
4. die Bumps werfen keine Schatten aufeinander
5. Oberflächennormalen auf der lichtabgewandten Seite eines Objektes können trotzdem zum Licht gewandt sein und erhalten dann fälschlicherweise Licht!

Displacement Mapping

Für jeden dieser Fehler gibt es mehr oder weniger aufwändige Hilfslösungen, korrekt ist es aber, die Oberfläche tatsächlich um die Bumphöhe zu verändern. Diese Methode nennt sich dann *Displacement Mapping*. Dabei werden die Oberflächenpunkte tatsächlich verschoben und es entsteht natürlich auch eine korrekte Silhouette. Dies ist aber natürlich viel aufwändiger zu implementieren und wird daher selten verwendet. Allerdings gibt es einen Trend dazu, diese Möglichkeit auf der Graphikkarte in Hardware zu unterstützen.

Kombination mehrerer Mappings



Eine sehr mächtige Methode ist es, mehrere Mappings zu kombinieren. Man nennt das im Falle der Texturen auch *Multitexturing*. Beispiele von kombinierbaren Texturen sind: Grundmuster, Beleuchtung, Verschmutzung, Unebenheiten, aber auch zum Beispiel Fotos plus Annotationen. Weiters kann man auch noch Environment Mapping und Displacement Mapping dazu kombinieren, man erhält dann Bilder wie das abgebildete.