Die Grundlagen des Texture Mapping

Sascha Vöhringer*

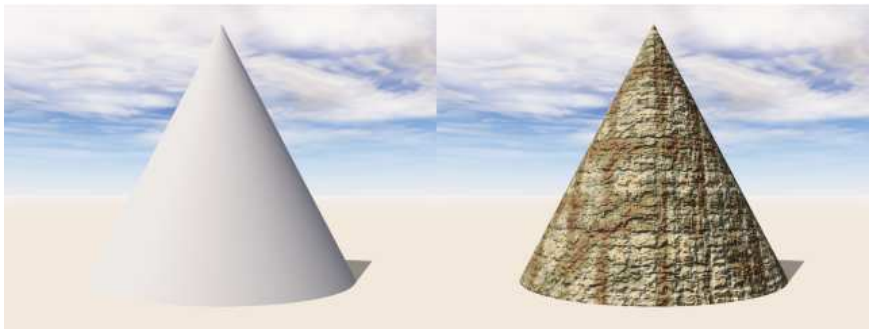

Abbildung 1: Zylinder ohne und mit Textur (aus vue6)

Zusammenfassung

In folgender Arbeit werden zusammenfassend die Grundlagen, die Funktionsweise und die Motivation für Texture Mapping erläutert.

In Abschnitt 1.) werde ich kurz erläutern, warum es so etwas wie Texturen gibt, wozu sie gut sind und für welche Zwecke man sie einsetzen kann. Auch werde ich die verschiedenen Arten von Texturen aufzeigen, wie diskrete Texturen auf der Basis von Bilder und prozeduralen Texturen, die auf mathematischen Funktionen basieren (siehe Abschnitt 2.2.1 und 2.2.2). In Abschnitt 2.3) wird Vorgang des Mappens anhand einer Kugel erläutert, auf die eine Weltkarte aufgebracht werden soll, um so einen Globus zu erstellen. Am Ende der Arbeit, in Abschnitt 2.4), werden wir das eher selten eingesetzte 3D-Textute Mapping und die Funktionsweise der Perlin-Noise und Turbulence-Funktion betrachten.

CR Categories: I.3.7 [Color, shading, shadowing, and texture]: Three-Dimensional Graphics and Realism—Computer Graphics Computing Methodologies

Keywords: texture mapping,Texturen,Texel, Parametrisierung,Projektionstransformation,nearest neighbor, Interpolation,Perlin Noise

1 Introduction

Im Gegensatz zu wirklich existierenden Objekten wirkten alle polygonialen Objekte eher glatt und Detaillos. Kratzer auf der Oberfläche, Spuren von Dreck, Fingerabdrücke und Reliefs waren nur durch rechenaufwendige geometrische Veränderung des Objekts realisierbar. Das explizite Wiedergeben von Oberflächen durch Modellierung ist jedoch oft zu aufwendig, Grafik-Beschleunigungs-Software schafft zwar oft Abhilfe, dennoch wurden im Laufe der Zeit einige Optimierungsverfahren in dieser Hinsicht entwickelt. Um Rechenzeit einzusparen, die Oberflächen aber dennnoch realistisch aussehen zu lassen, Unebenheiten zu simulieren, und vieles mehr kommt das von Edwin Catmull entwickelte Texture Mapping zum Einsatz (Abb.1)

Texture Mapping ist eine effiziente Methode um einer Oberfläche mehr Realismus und Details hinzuzufügen, ohne dass weitere Geometrie erzeugt werden muss. Anschaulich kann man das Verfahren wie folgt beschreiben: Ein Poster wird an eine weiße Wand geklebt, dort entsteht nun möglicherweise ein komplexes, räumliches Bild, ohne dass man die Oberfläche der Wand geometrisch verändert hat.

Eine Texture Map kann ein- zwei- oder dreidimensional sein und kann sowohl als mathematische Funktion als auch ein Array repräsentiert werden. Die Funktion, bzw. das Bild (repräsentiert durch ein Array) wird *Texture Map* genannt, der Prozess, diese Funktion auf eine meist 3-Dimensionale Oberfläche zu legen nennt man *texture mapping*. Eine dreidimensionale Textur kann beispielsweise Wolken oder Murmeln simulieren, man nennt sie *solid textures* oder *volume textures*. 2D Texturen werden als image textures bezeichnet. Den Unterschied zwischen 2D und 3D Texture Mapping werde ich später in dieser Arbeit noch genauer erläutern. Für unsere Zwecke werden jedoch meist 2D-Texturen betrachtet, also 2-dimensionale Arrays.

Jedes Bild dass dazu benutzt wird um die Objekt-Beschaffenheit zu beeinflussen nennt man Textur. Mit verschiedenen Techniken, wie Bump-Mapping kann die Textur noch weiter verfeinert werden, indem zB. Höhenunterschiede der Oberfläche simuliert werden.

2 Exposition

Die Quelle, welche die Bilddaten beinhaltet die auf eine Oberfläche abgebildet werden sollen, wird als Textur (engl. texture) bezeichnet. Sie besteht aus einer Ansammlung von Texeln (Einzelne Pixel der Texture Map), welche die kleinsten Informationseinheiten darstellen. Ein Texel kann neben Farbwerten, die der Substitution von Pixeln dienen, auch Informationen über Materialeigenschaften

*e-mail: sascha.voehringer@uni-ulm.de

(wie z.B. Transparenz oder Normalenvektoren), die beim Rendern berücksichtigt werden, enthalten.

Wir bezeichnen im Folgenden die Koordinaten der Textur als u und v, die Koordinaten im Objektraum als (x_0, y_0, z_0) und die Bildschirmkoordinaten als (x,y). Die Funktion c(u,v) liefert die Farbinformation des Texels an der Stelle (u,v). u und v bilden den (u,v)-Raum. Im dreidimensionalen kommt noch eine zusätzliche Koordinate hinzu, man spricht dann vom (u,v,w)-Raum

2.1) 2D Texture Mapping

Der Vorgang des Texture Mappings im Zweidimensionalen lässt sich in 2 Schritte unterteilen:

- Jeder Bestandteil der Datenquelle, also jedes Texel wird zunächst auf eine 3-Dimensionale Oberfläche projiziert, d.h. jedem Punkt des Koordinatensystems der Quelle wird ein räumlicher Punkt der Oberfläche zugewiesen.
  (siehe Abb.2, Oberflächenparametrisierung)

- Die 3D-Koordinate wird auf den Bildschirm projiziert, wo sie als Pixel gerendert wird. (siehe Abb.2, Projektion)

Dies nennt man Vorwärts Mapping, oder auch Parametrisierung. Die Parametrisierung verbindet alle Punkte einer Texturebene mit Punkten auf einer Oberfläche.

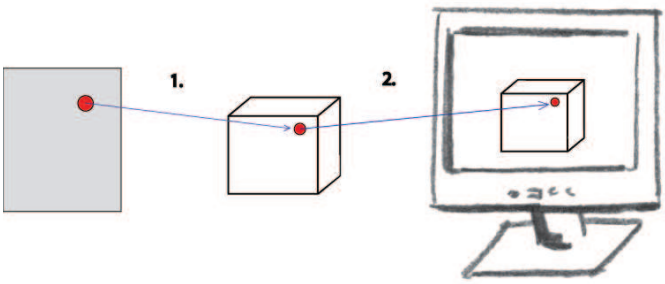

Abbildung 2: Transformation eines Pixel

In der Praxis wird dieser Vorgang jedoch meist umgekehrt realisiert, da dies algorithmisch effizienter zu realisieren ist, man nennt diesen Vorgang Rückwarts-Mapping oder auch Projektionstransformation: Eine Oberfläche wird auf den Bildschirm projiziert und für jeden Pixel wird die Position in der Bildquelle ermittelt, an der sich die Farbinformation c(u,v) für den darzustellenden Punkt befindet (siehe auch Abb.3)

Um eine Textur also erfolgreich auf eine Oberfläche zu projizieren, müssen mehrere Rechenschritte durchlaufen werden, um die Koordinaten richtig zu transformieren.

Wir gehen davon aus,dass ausschließlich Dreiecke texturiert werden, was keinesfalls die Beschaffenheit der zu modelierenden Objekte einschränkt, da sich alle Polygone in Dreiecke zerlegen lassen. Dreiecke besitzen den Vorteil dass sie stehts planar (plättbar) sind, ausserdem arbeiten alle Echtzeit-Rendering-Maschinen auf der Basis von Dreiecken. Betrachten wir zunächst die 2 möglichen Arten einer Textur, nämlich die auf der Basis eines bestehenden Bildes oder einer Grafik und die auf Basis einer mathematischen Funktion.

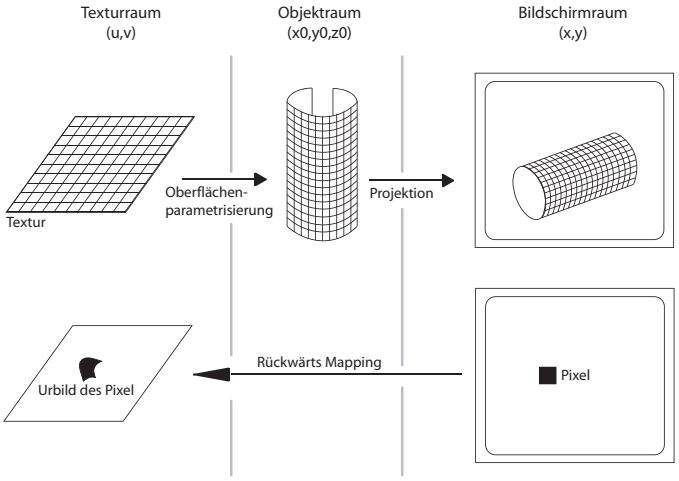

Abbildung 3: Theoretischer und praktischer Mapping-Vorgang

2.2.1) Diskrete Textur / Texture Arrays

Ein Weg um Texturen räumlich zu spezifizieren ist es, ein 3D-Array mit Farbwerten zu speichern und jedem dieser Werte eine räumliche Position zuzuordnen. Dies ist besonders einfach, da die Transformation in den Objektraum entfällt, da jedem (x_0, y_0, z_0) direkt ein Punkt aus dem uvw-Raum zugeordnet werden kann. Betrachten wir dies zunächst für 2D-Arrays im 2D-Raum.

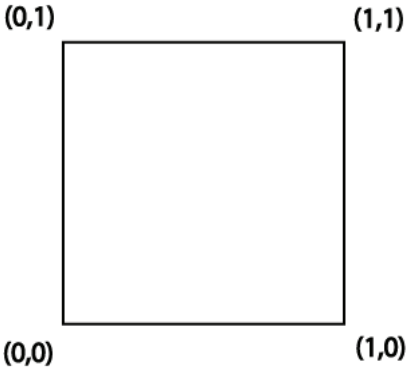

Abbildung 4: Das Einheitsquadrat

In der Anwendung werden Texel nicht direkt, sondern mittels zweier Koordinaten $u, v \in 0, 1$ adressiert. Die Textur, die auf das Objekt aufgebracht werden soll, hat die Breite n_x und die Höhe n_y. Jeder Kante eines Polygons muss nun eine Texturkoordinate t = (u,v) zugewiesen werden. Zuerst wird der Integer Anteil des Punktes (u,v) entfernt, so dass er im Einheitsquadrat liegt (Abb.4). Die uv-Koordinate (0,0) entspricht der unteren linken Ecke des Bildes, die uv-Koordinate (1,1) der oberen rechten Ecke. uv-Werte größer 1 und kleiner 0 sind möglich und führen zu Randwiederholungseffekten des Bildes. Das hat den Effekt, dass die uv-Ebene sozusagen mit Kopien der nun quadratischen Textur gekachelt wird (Abb.6). Die Werte für u und v im Bereich 0,0 und 1,0 bestimmen die äußeren Bereiche einer Textur.

Dies hat möglicherweise den unerwünschten Effekt, dass sogenannte Kachelmuster auftreten, sobald die Textur mehrfach auf eine Oberfläche aufgelegt wird, falls die Kanten der Textur nicht ineinander übergehen.

Bei der einfachen Variante des Texture Mapping werden die in den

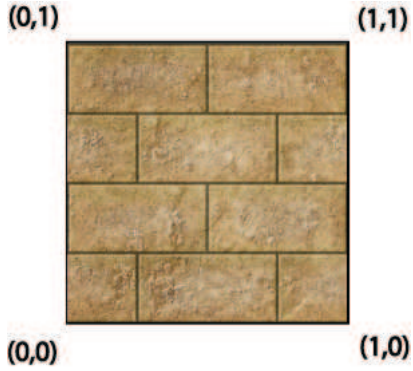

Abbildung 5: Textur im Einheitsquadrat

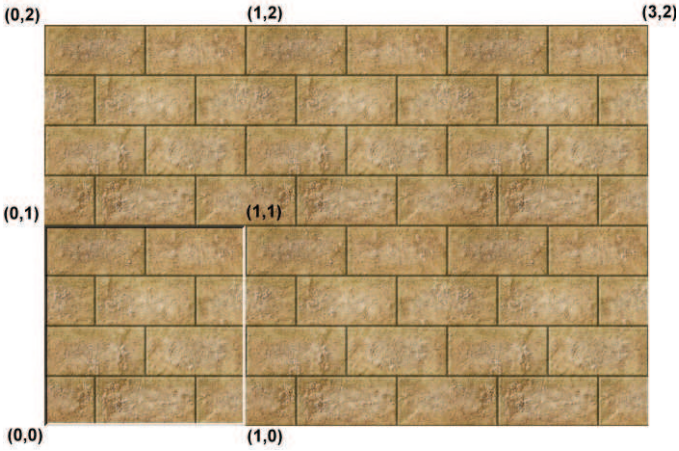

Abbildung 6: uv-Ebene

Objektraum transformierten Texturkoordinaten entlang der Randlinien eines Polygons *linear Interpoliert*. Jedem Pixel wird der Farbwert des, zur jeweiligen interpolierten (u,v)-Koordinate gehörenden, Texels übernommen.

Dieses Verfahren kann zu unbefriedigenden Resultaten führen, falls die Ausdehnung des Objekts in Sichtrichtung größer wird, da die Interpolation dies nicht berücksichtigt. In diesem Fall muss eine Perspektivenkorrektur durchgeführt werden, die in der Arbeit von Volker Börner genauer erläutert wird.

Bisherige Verfahren nehmen an, dass jedem Pixel exakt ein Texel zugeordnet werden kann, was im Allgemeinen nicht der Fall ist. Meistens liegen die Texturkoordinaten eines Pixels sogar zwischen mehreren Texeln. Nun muss entschieden werden, welchen Farbwert der Pixel nun endgültig erhält, wobei geeignete Interpolationsverfahren zum Einsatz kommen. Die einfachste ist hierbei die *'nearest neighbor'* oder auch *'point sampling'* Methode: Jeden Texel als eine konstante rechteckige Kachel zu betrachten. Dieser Algorithmus sorgt dafür, dass bei der Umrechnung von Texturkoordinaten $\in [0,1]$ in diskrete Pixel korrekt gerundet wird. Dabei wird dasjenige Texel, das am nächsten an einem Bildpunkt liegt zur Darstellung gewählt. Die Farbinformation wird durch $c(u,v) = c_i j$ berechnet, wobei c(u,v) die Farbe der Textur an der Stelle (u,v) und $c_i j$ die Farbe des Pixels mit entsprechendem Indizé ist:

$$i = \lfloor un_x \rfloor$$
$$j = \lfloor vn_y \rfloor$$

Die Indizes beginnen bei (i,j)=(0,0)

Nearest Neighbor Sampling wählt also für einen Pixel von allen möglichen Texeln den Wahrscheinlichsten aus. Das Verfahren ist einfach zu realisieren, bietet jedoch nur geringen Nutzen, da stets nur ein Texel für einen Pixel in Betracht kommt. Auch ist diese Methode nur geeignet, wenn die Textur während einer Transformation nicht in ihrer Größe verändert wird, da sonst schon erwähnte, ungewollte Kachelmuster und Aliasingeffekte auftreten können.

Um eine weichere Textur zu erreichen und Aliasingeffekte beim Verkleinern oder Vergrößern der Textur zu verhindern, werden verschiedene Filtermechanismen benutzt, darunter fällt auch die *'bilineare Interpolation'* oder *'bilineares Filtern'*. Bei dieser Methode werden für ein einzelnes Pixel vier umliegende Texelwerte berücksichtigt, woraus das gewichtete Mittel berechnet wird und der resultierende Farbwert dargestellt wird.

c(u,v) berechnet sich wie folgt:

$$\begin{aligned} c(u,v) = &(1-u')(1-v')c_{ij} \\ &+u'(1-v')c_{(i+1)j} \\ &+(1-u')v'c_{i(j+1)} \\ &+u'v'c_{(i+1)(j+1)} \end{aligned}$$

mit

$$u' = n_x u - \lfloor n_x u \rfloor$$
$$v' = n_y v - \lfloor n_y v \rfloor$$

Unstetigkeiten der Intensität die durch der Ableitungen enstehen kann sichtbare Linien hervorrufen, um dies zu umgehen kann als dritte Möglichkeit die Hermite Interpolation verwendet werden

$$\begin{aligned} c(u,v) = &(1-u'')(1-v'')c_{ij} \\ &+u''(1-v'')c_{(i+1)j} \\ &+(1-u'')v''c_{i(j+1)} \\ &+u''v''c_{(i+1)(j+1)} \end{aligned}$$

mit

$$u'' = 3(u')^2 - 2(u')^3$$
$$u'' = 3(v')^2 - 2(v')^3$$

Im 3-dimensionalen geht man analog vor. Der Unterschied ist, dass man statt eines 2D-Arrays ein 3D-Array benutzt,

$$\begin{aligned} c(u,v,w) = &(1-u')(1-v')(1-w')c_{ijk} \\ &+u'(1-v')(1-w')c_{(i+1)jk} \\ &+(1-u')v'(1-w')c_{i(j+1)k} \\ &+(1-u')(1-v')w'c_{ij(k+1)} \\ &+u'v'(1-w')c_{(i+1)(j+1)k} \\ &+u'(1-v')w'c_{(i+1)j(k+1)} \\ &+(1-u')v'w'c_{i(j+1)(k+1)} \\ &+u'v'w'c_{(i+1)(j+1)(k+1)} \end{aligned}$$

mit

$$u' = n_x u - \lfloor n_x u \rfloor$$
$$v' = n_y v - \lfloor n_y v \rfloor$$
$$w' = n_z w - \lfloor n_z w \rfloor$$

Eine weitere Technik die an dieser Stelle noch zu erwähnen wäre, ist das MIP-Mapping, welches angewandt wird, falls der Rasterabstand der Pixel kleiner oder größer als der der Texel ist, also einem Pixel höchstens ein Texel zugeordnet werden kann oder ein Pixel einen ganzen Bereich der Textur überdeckt. MIP-Maps enthalten

neben der Originaltextur noch eine Folge weiterer *Texturkopien* mit abnehmder Auflösung. So kann das Pixel-Texel-Verhältnis besser bestimmt werden. Die Texturkkopien werden dann wie die Originaltextur verwendet.

2.2.2) Prozedurale Textur

Prozedurale Texturen unterscheiden sich von diskreten Texturen weitestgehend darin, dass sie nicht photographiert oder gezeichnet wurden, sondern vorab oder sogar live im Computer durch Komposition von (meistens nicht linearen und stochastischen) Funktionen berechnet werden. Unerwünschte Kacheleffekte werden so vermieden. Im Echtzeit Rendering bedeutet dies jedoch Einbuße der Geschwindigkeit, da realistische Bilder auch sehr komplexe Algorithmen erfordern. Im Kapitel 2.4) wird das Verfahren anhand der Perlin Noise genauer erläutert.

Wir betrachten im Folgenden die gebräuchlichen diskreten 2D Texturen

2.3) Mapping anhand einer Sphere

Das Koordinatensystem (u,v) wird gebildet, so dass es mit Punkten eines Objekts im Objektraum assoziiert werden kann. Als Grundlage muss das Koordinatensystem der Textur dem Einheitsquadrat entsprechen, wie schon weiter oben beschrieben

$$(u,v) \in [0,1]$$

Für alle (u,v) ausserhalb dieses Quadrates werden nur die Bruchstellen der Koordinate benutzt. Als Resultat bekommt man, wie weiter oben beschrieben, die gekachelte (uv)-Ebene.

Um beispielsweise eine Kugel (Abb.8) mit einer Textur (Abb.7) zu überziehen, erinnern wir uns an das Kugelkoordinatensystem. Für eine Kugel mit Radius r und Mittelpunkt (c_x, c_y, c_z), lautet die parametrische Gleichung einer Kugel :

$$\begin{aligned} x &= x_c + r\cos(\Phi)\sin(\Theta) \\ y &= y_c + r\sin(\Phi)\sin(\Theta) \\ z &= z_c + r\cos(\Theta) \end{aligned}$$

Wobei $0 \leq \Theta \leq \Pi$ und $0 \leq \Phi \leq 2\Pi$

Φ und Θ seien nun:

$$\begin{aligned} \Theta &= \arccos\tfrac{z-z_c}{r} \\ \Phi &= \arctan 2(y-y_c, x-x_c) \end{aligned}$$

(arctan2(a,b) berechnet den arctan von a/b)

Nun können wir in das Koordinatensystem (u,v) konvertieren:

$$\begin{aligned} u &= \tfrac{\Phi}{2\Pi} \\ v &= \tfrac{\Pi-\Theta}{\Pi} \end{aligned}$$

Nun, da wir unser Objekt mit einer Textur belegt haben (Abb.9), müssen wir die Objektkoordinaten noch in Bildschirmkoordinaten umrechnen. Es sei P eine 3D-Koordinate und S die Bildschirmkoordinate:

$$\begin{aligned} S_x &= distance\tfrac{P_x}{P_z} \\ S_y &= distance\tfrac{P_y}{P_z} \end{aligned}$$

Die Konstante *distance* simuliert die Distanz vom Betrachter zum Bildschirm und bestimmt damit das Sichtfeld. Nachdem die Dreieckspunkte in Bildschirmpunkte transformiert wurden, wird die

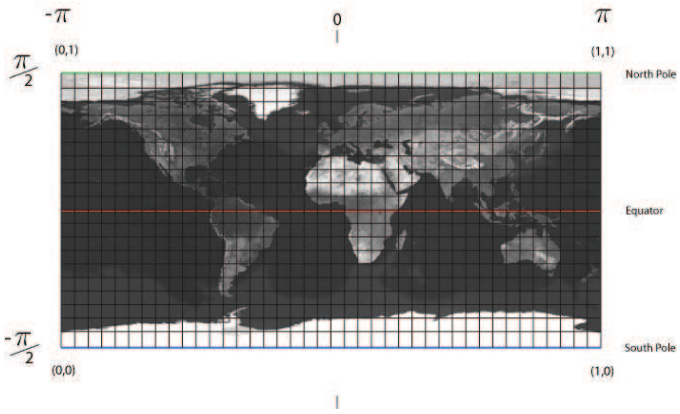

Abbildung 7: Textur eines Erdballs mit Liniensegmenten

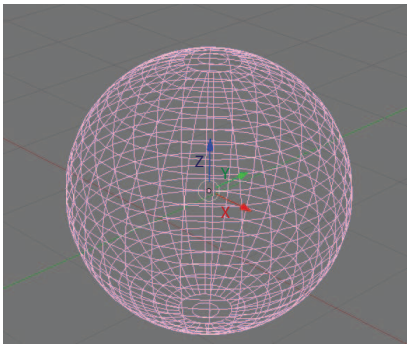

Abbildung 8: Drahtgitternetz einer Sphere (aus Blender)

projizierte Fläche mit horizontalen Linien von oben nach unten aufgefüllt. Dies geschieht mit diversen Rasteralgorithmen, wie zB. den Bresenham-Algorithmus, oder auch *'Midpoint-Algorithmus'*, von 1965. Dieser und andere grundlegende Algorithmen rastern Linien nur einfarbig. Eine bessere Darstellung mit mehreren Farbabstufungen ergibt sich bei fortgeschrittenen Verfahren, die Antialiasing (Kantenglättung) unterstützen.

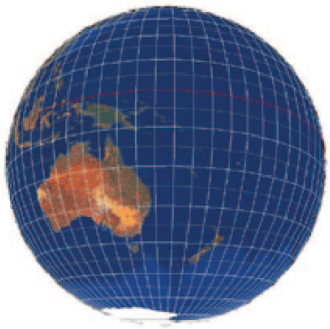

Abbildung 9: fertig texturierte Kugel

2.4) 3D Texture Mapping

2.4.1)Volumentexturen

volumetrische Texturen haben eine dreidimensionale Texturdomäne, haben also zusätzlich eine w-Koordinate, welche die Tiefe angibt.

$$(u,v,w) \in [0,1]^3$$

Ein Element einer volumetrischen Textur wird Voxel genannt.

Diese Texturen können direkt auf Oberflächen angewendet werden, wodurch die Umrechnung vom Texturraum in den Objektraum entfällt, da jedem Punkt des Objekt (x,y,z) direkt ein Voxel zugeordnet werden kann und so auch ungewünschte Verzerrungen nicht entstehen. Der Vorteil liegt also ganz klar bei dem geringen Transformationsaufwand. Eine 3D-Textur verbraucht jedoch viel mehr Speicher als eine einfache 2D-Textur, was teilweise eine schlechte Performance hervorruft.

2.4.2) 3-dimensionale Prozedurale Texturen

Wie bereits besprochen, können die Punkte eines 3D-Arrays direkt auf die 3D Oberfläche projiziert werden, so wie 2D-Arrays auf 2D Oberflächen projiziert werden können.

Über einfache mathematische Funktionen Texturen und Muster zu erstellen ist nicht allzu komplex, jedoch sind Diese dann nicht sehr wirklichkeitsgetreu. Einige Funktionen, die pseudozufällige Muster erzeugen, nennt man Noise-Funktionen (Abb.10). Auch diese Funktionen können isoliert keine wirklichkeitsgetreuen, realistisch wirkenden Muster erzeugen, sie wirken immer noch zu 'künstlich' und zu 'linear'. Legt man diese Funktionen aber sozusagen übereinander, können zufällige, realistisch wirkende Muster entstehen. Hier kommt dann die sogenannte 'solid noise' oder auch 'Perlin Noise' zum Einsatz. Da Perlin Noise darauf basiert, scheinbar zufällige Funktionen mit weichen Übergängen (die Steigung kann einen gewissen Wert nicht überschreiten) zu benutzen, eignet sie sich sehr gut zur Simulation verschiedenster, natürlicher Gebilde, wie die Oberfläche eines Ei's, oder auch Wolken.

Um eine Perlin Noise anzuwenden, benötigt man mehrere Rauschfunktionen (Abb.10) mit verschiedenen Werten. Eine Rauschfunktion ist eine quasi randomisierte Funktion. Sie liefert einen zufälligen Wert zwischen -1 und 1, jedoch bei gleicher Eingabe, im Gegensatz zu anderen Zufalls-Funktionen (zum Beispiel Math.random in Java), wieder den gleichen Wert. Würde man für jeden Pixel lediglich eine Zufallszahl berechnen, bekäme man nichts weiter als eine Art 'Weißes Rauschen', wie man es aus dem Fernsehen kennt.

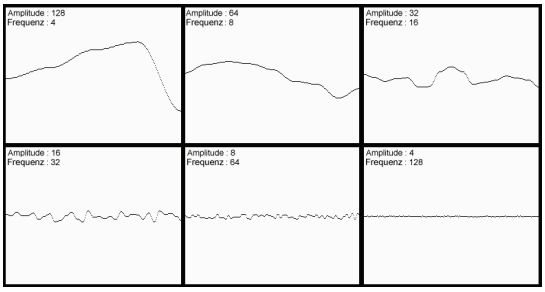

Abbildung 10: Mehrere erzeugte Rauschfunktionen

Da diese Rauschfunktionen noch nicht echt wirken, werden sie summiert und man erhält eine Perlin Noise. Dazu wird die Basisformel von Perlin benötigt:

$$n(x,y,z) = \sum_{i=\lfloor x \rfloor}^{\lfloor x \rfloor+1} \sum_{j=\lfloor y \rfloor}^{\lfloor y \rfloor+1} \sum_{k=\lfloor z \rfloor}^{\lfloor z \rfloor+1} \Omega_{ijk}(x-i, y-j, z-k)$$

Das nun entstandene Muster sieht schon eher aus wie zB. Eine Gebirgskette (Abb.11). Auf eine Kugel angewendet, entsteht ein weiches, zufälliges Muster (Abb.12)

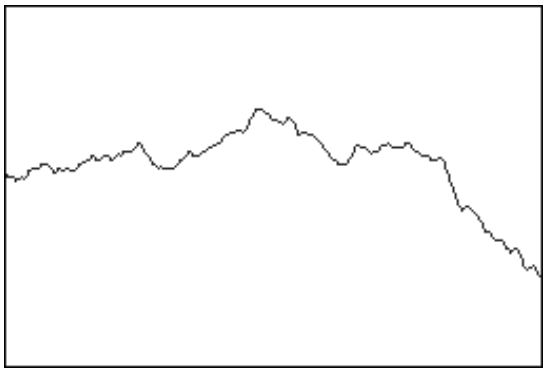

Abbildung 11: Eine mögliche Perlin Funktion

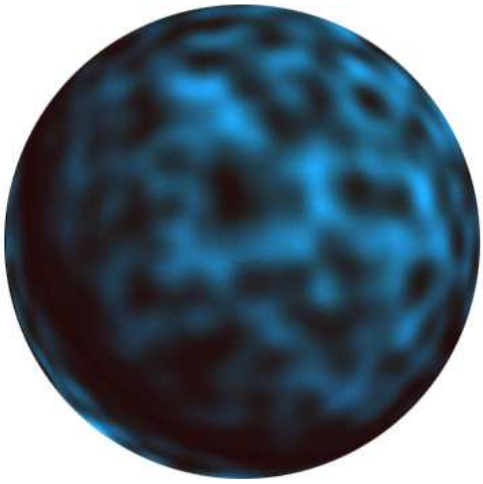

Abbildung 12: Eine Perlin Funktion auf eine Sphere angewendet

Erwähnt sei an dieser Stelle noch die Turbulence Funktion:

Die Turbulence-Funktionen erzeugt wie der Name schon sagt einen turbulenten skalaren Wert und wird unter Verwendung spezieller Kombinationen mathemathischer Funktionen mit Noise-Werten generiert. Eine eindimensionale Variante sieht wie folgt aus:

$$n_t(x) = \sum i \frac{|n(2^i x)|}{2^i}$$

Diese Funktion stapelt wiederholt skalierte Kopien der Noise-Funktion, um den gewünschten Effekt zu erreichen.

3 Schlussfolgerung

Texture Mapping eignet sich hervorragend dazu, eine Oberfläche realistisch wirken zu lassen, ohne die Geometrie des Objekts zu

verändern, somit wird Rechenzeit eingespart. Durch diverse Filteralgorithmen kann auch in Abhängigkeit vom Blickwinkel ein gutes Ergebnis erzielt werden. Volumetrische Texturen sind zwar geschickter zu handhaben, da die Umrechnung in den Objektraum, und somit unerwünschte Effekte entfallen, momentan jedoch noch etwas zu Rechenzeitaufwendig, weshalb in der Praxis noch immer 2D-Texturen zum Einsatz kommen. Texturen können desweiteren entweder diskret oder prozedural sein, Eine diskrete Textur ist ein Foto oder ein Bild und eine prozedurale Textur ist eine mathematische Funktion. Der Vorgang des Mappens sieht wie folgt aus: Eine Textur wird in Objektkoordinaten umgerechnet, einem Texel wird, je nach Prozess, ein Pixel zugewiesen und das texturierte Objekt in Bildkoordinaten umgerechnet. Gerendert wird das Objekt von oben nach unten durch horizontale Linien. In der Praxis kommt das Verfahren der Projektionstransformation zum Einsatz, dh. Vereinfacht: Ein Objekt wird auf dem Bildschirm angezeigt und die zu mappende Textur wird darübergelegt. Es gibt noch viele Verfahren um eine Textur noch realistischer wirken zu lassen, wie etwa das 'Bump Mapping', welches Oberflächenunebenheiten simuliert, was aber in der Ausarbeitung von Volker Börner unter Anderem genauer erläutert wird.

Literatur

KÖGLER, S. 2002. *Texture Mapping in echtzeitberechneten virtuellen Umgebungen*. Master's thesis, Universität Hamburg.

PERLIN, K. Making noise. GDCHardCore, Decembre.

PETER SHIRLEY, MICHAEL ASHIKHMIN, M. G. S. R. M. E. R. K. S. W. B. T., AND WILLEMSEN, P. 2005. *Fundamentals of computer graphics*. A. K. Peters.