Auch das noch!

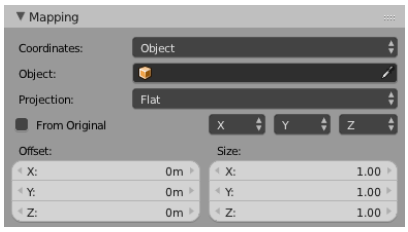
Verwirrendes Mapping

Autor: Uwe Gleiß, Franz-Ludwig-Gymnasium Bamberg, Computergrafikgruppe (CoGra-FLG) • Kontakt: cogra-flg@web.de

Problem

Mappingeinstellungen

Beim Arbeiten mit Texturen in Blender Render stolpert man früher oder später über diese Einstellungen:



Gefürchtete Mappingeinstellungen

Nachfolgend wird (wieder einmal) versucht, die damit zusammenhängende Verwirrung zu beheben (auch für mein eigenes zukünftiges Gehirn). Dabei werde ich um etwas Mathematik (es ist anschauliche Geometrie, tut also nicht so weh) nicht ganz herum kommen.

Gründe für die Verwirrung

Vielleicht helfen die folgenden Anmerkungen schon über manchen Stolperstein hinweg:[1]

- Die Einstellungen unter Mapping gehören nicht zur Textur, sondern zum Eintrag in der Texturliste. Verwendet man eine Textur an anderer Stelle noch mal, so wird das Mapping nicht übernommen (neben der Liste mit dem dunklen Dreieck kann man die Einstellungen samt Mapping kopieren bzw. einfügen).
- Coordinates sind immer frei wählbar. Das bedeutet noch lange nicht, dass alle Einstellungen Sinn ergeben.
- Projection ist nur für brettflache Bildtexturen von Bedeutung (mehr dazu unten).
- Die drei Koordinatenkästchen unter Projection sind bestens für Verwirrung geeignet. Dabei ist das Prinzip eiiiigentlich eindeutig und vielleicht sogar simpel.

Kern des Problems

Eine Textur hat Koordinaten (aber nicht immer die gleiche Sorte) und unser Objekt im Raum hat Koordinaten (auch nicht immer die selben). Das Mapping hat den Zweck, den Transfer von der Texturkoordinaten zu Objektkoordinaten zu leisten. Trotz teils unterschiedlicher Daten steuert man das in Blender immer mit den gleichen Steuerelementen, was einem schon mal einen Knoten ins Gehirn zaubern kann.[2]

[1] Dies ist nicht als Meckern an Blender zu verstehen – vielleicht ließe sich das besser lösen, aber das ist immer auch Geschmacksache.

[2] Gut, das war jetzt ein klein wenig gemeckert.

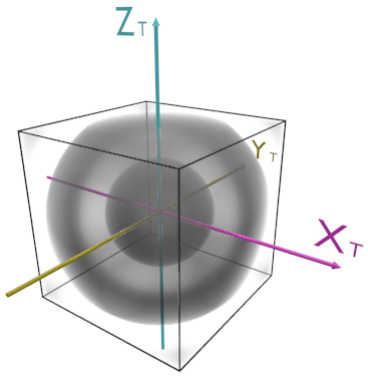
Erklärungsversuch

Texturkoordinaten

Damit man für einen Punkt auf der Oberfläche des Objekts einen Punkt in der Textur aussuchen kann, benötigen beide ein Koordinatensystem. Fangen wir mit Texturkoordinaten an, die wir zur Unterscheidung zu Objektkoordinaten mit X_T, Y_T und Z_T bezeichnen. Der einfachste Fall, diese festzulegen ist: Man hat sie schon:

3D Texturen

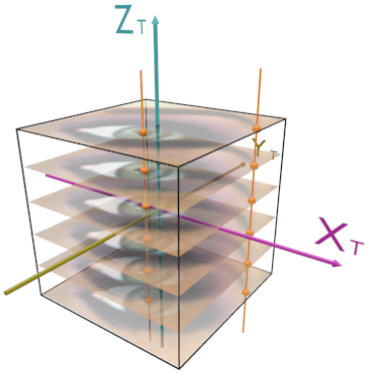
Eine 3D Textur ist eine Formel, die jeder Kombination aus X_T, Y_T und Z_T einen Zahlenwert zuordnet. Die Zahlen kann man z.B. Farben zuordnen, so dass ein Muster entsteht. Als Beispiel nehmen wir hier mal einander umgebende Kugelschalen ähnlich den Schichten in einer Zwiebel (Wood Textur).



Bei solchen Texturen macht eine Projektion keinen Sinn.[3] Was im Mapping bei Projection eingestellt ist, ist egal. Die meisten Texturtypen in Blender gehören zu dieser Kategorie. Im Einzelnen sind das Blend, Clouds, Distorted Noise, Magic, Marble, Musgrave, Noise, Stucci, Voronoi und Wood. Auch Point Density und Voxel Data funktionieren prinzipiell nach diesem Prinzip, haben aber gewisse Eigenheiten (die hier größtenteils erst mal außen vor bleiben).

2D Texturen

Meist handelt es sich um Bilder, die ja nur zwei Koordinaten besitzen. Für die weitere Verarbeitung werden daraus immer drei Koordinaten gebildet. Dieser Schritt, die Projection, erzeugt erst die Texturkoordinaten X_T, Y_T und Z_T. Wo im Texturraum welche Farbe aus dem Bild landet, veranschaulichen hoffentlich die nachfolgenden Bilder.[4]



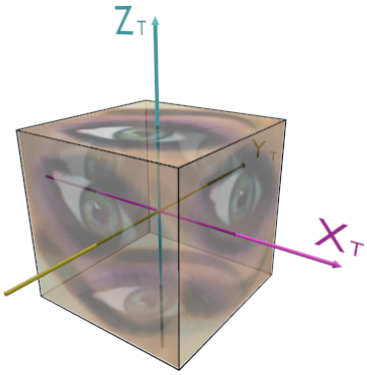
Flat

Der Texturraum wird mit Farbe gefüllt, als würde sich das Bild als Ebene in Z_T Richtung bewegen und dabei einen Kondensstreifen aus Farbe zurücklassen. Direkte Folge: Die Z_T Koordinate ist für die Färbung bedeutungslos, da alle Punkte, die „übereinander“ liegen, sowieso gleich gefärbt sind (die orangen Linien im Bild sollen das verdeutlichen).

---

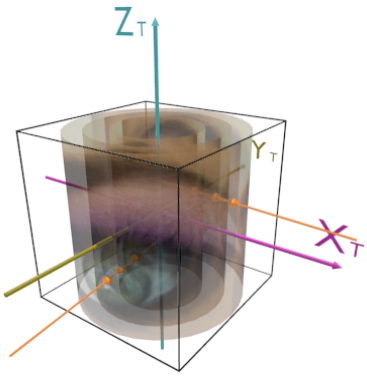
[3] Das stimmt so nicht ganz. Andere 3D Programme können eine Scheibe aus einer 3D Textur verarbeiten wie ein Bild und das kann projiziert werden.

[4] Das Bild des Auges stammt von Courtney Rhode, gefunden auf Flickr und dort verfügbar unter CC-Lizenz by.



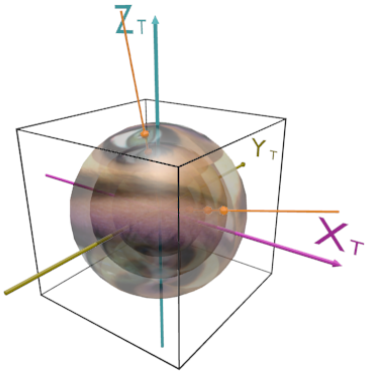
Cube

Cube erscheint auf den ersten Blick simpel (und ist es in der Anwendung auch), die genaue Spielregel für das Bestimmen einer Farbe ist aber nicht so einfach, denn sie besteht aus drei Projektionen wie bei Flat. Eine entlang der Z_T Achse wie bei Flat, eine entlang der X_T Achse und eine entlang Y_T.
Das ergäbe eigentlich für jeden Raumpunkt drei verschiedene Farben und damit Chaos, darum muss zusätzlich entschieden werden, welche der drei Projektionen zum Einsatz kommt. Dazu berechnet Blender, welche der drei Achsen „am senkrechtesten" auf der betroffenen Fläche steht. Die entsprechende Projektion gewinnt.



Tube

Stellen wir uns eine Folien unterschiedlicher Breite aber immer gleicher Höhe vor. Das Bild wird für jede Folie so horizontal gestreckt, dass es in der Breite genau auf passt. Diese Folien Wickeln wir jetzt eine um die andere.
Folge: Für die Farbe eines Punktes ist nur die Z_T Koordinate und der Abstand von der Z_T Achse entscheidend. Alle Punkte in gleicher „Höhe" und „Himmelsrichtung" (X_T als Nord-Süd und Y_T als Ost-West) erhalten die selbe Farbe.



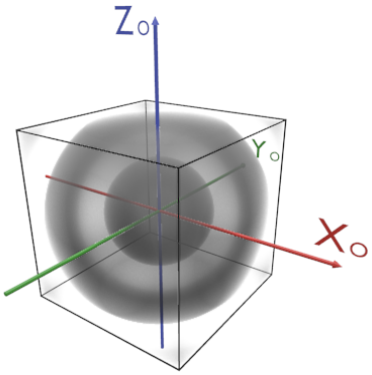
Sphere

Das gleiche Prinzip wie bei Tube, nur dass wir diesmal die Folie zur Kugel biegen. Das führt unvermeidlich an den Polen (entlang der Z_T Achse) zu Verzerrungen.
Folge: Die Farbe eines Punktes im Texturraum hängt nur davon ab, in welche Richtung man vom Ursprung aus zu ihm blickt. Der Abstand vom Ursprung spielt keine Rolle.

2D Texturen sind Image or Movie und in gewisser Weise auch Environment Map und Ocean. Wir beschränken uns hier erst mal auf Erstere.

Koordinatenzuordnung

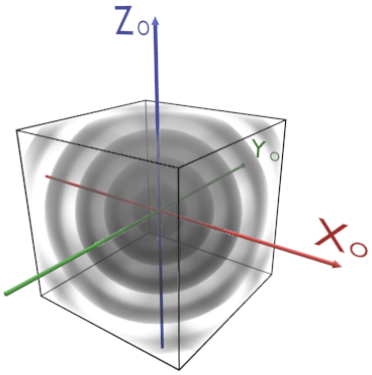
Widmen wir uns Koordinaten auf einem Objekt. Gemeinerweise sind das genau die, die in Blender neben Coordinates in einer Liste ausgewählt werden können, die mit Texture Coordinates überschrieben ist. Liest man diese eher als: „Koordinaten des Objektes, die in den Texturraum übertragen werden sollen“, dann ergibt das Ganze (hoffentlich) mehr Sinn. Nachfolgend nennen wir diese Koordinaten X_O, Y_O und Z_O, auch wenn es sich in diversen Fällen gar nicht um X, Y und Z im klassischen Sinne handelt.
Bevor diese Tatsache Panik auslöst, sehen wir uns den einfachsten Fall an und spielen an diesem auch gleich durch, was die ominösen drei Koordinatenkästchen für eine Rolle spielen. Dabei gehen wir zunächst von einer mathematischen 3D Textur aus (weil Mappingmethoden wie Cube die Sache wenig übersichtlich machen).



Generated

Jedes Objekt besitzt sein eigenes privates Koordinatensystem, das bei seiner Generierung mit erzeugt wird. Wenn dieses nicht verformt wurde[5] und die Werte unter Mapping-Size alle auf 1 stehen, dann sind Texturkoordinaten und Objektkoordinaten identisch.
Blicken wir aber gleich ein wenig mehr unter die Haube: Blender benötigt eine (oder mehr) Formeln, mit denen aus bekannten Objektkoordinaten X_O, Y_O und Z_O die gewünschte Kombination aus Texturkoordinaten X_T, Y_T und Z_T berechnet werden kann.
In diesem einfachsten Fall sind die Formeln schlicht und einfach:

$X_T = X_O$; $Y_T = Y_O$; $Z_T = Z_O$

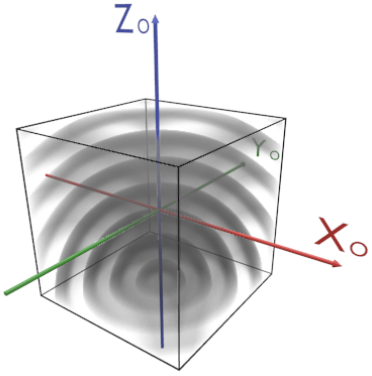


Size und doch nicht Size

Variieren wir die obigen Formeln ein klein wenig:

$X_T = 2 \cdot X_O$; $Y_T = 2 \cdot Y_O$; $Z_T = 2 \cdot Z_O$

Der Punkt (1|3|2) des Objekts wird also zu (2|6|4) im Textursystem. Bei gleicher Größe des Objekts wird dadurch ein größerer Raum der Textur abgetastet. Die Folge ist: Das Muster schrumpft.
Jetzt ist hoffentlich klar, warum die Werte unter Mapping-Size anscheinend genau falsch herum funktionieren. Soll das Muster also in X-Richtung auf das 4-fache gedehnt werden, dann müssen die Werte bei Size …[6]



Rutsch mal nen Schluck

Auch wenn man durch Änderung des Offset die Textur verschieben möchte, muss man umgekehrt denken. Setzen wir zusätzlich zu 2 bei Size auch noch Z bei Offset auf 1, sehen die neuen Formeln so aus:

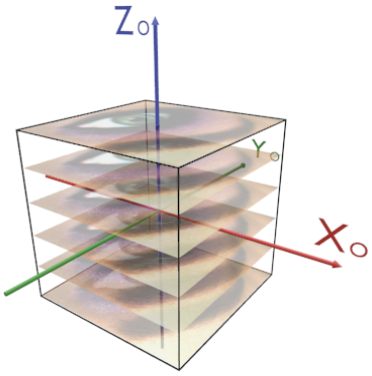
$X_T = 2 \cdot X_O$; $Y_T = 2 \cdot Y_O$; $Z_T = 2 \cdot (Z_O + 1)$

Der Ursprung (0|0|-1) der Objektkoordinaten wird damit zu (0|0|0) im Texturraum. Damit liegt die Mitte der Textur nun tiefer und nicht, wie vielleicht erwartet, höher.

---

[5] Skalierungen im Objektmodus verändern nicht das Objekt, sondern dehnen oder stauchen sein Koordinatensystem. Das kann man in den Properties des 3D Bereichs unter Size (nicht Dimensions) erkennen. Steht da mal nicht drei mal 1, dann kann man das mit Ctrl+A und dann Scale wieder auf 1 setzen, ohne die Form des Objekts zu verändern.

[6] … natürlich 0,25; 1 und 1 sein – da sind Sie aber sicher selbst drauf gekommen.

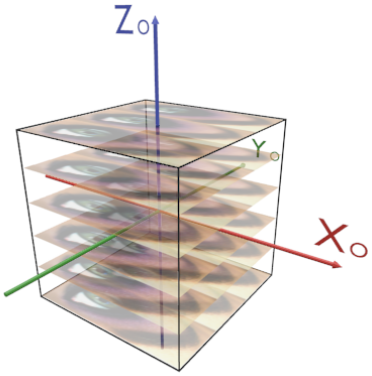


Magische Kästchen

Die drei Kästchen unter der Einstellung für Projection stehen für die drei Texturkoordinaten X_T (links), Y_T (mitte) und Z_T (rechts). Die Buchstaben, die man in den Kästchen eintragen kann, stehen dagegen für die Objektkoordinaten X_O, Y_O und Z_O, die in der jeweiligen Formel verwendet werden sollen. Tauschen wir bei einer Bildtextur mit Projektion Flat (S. 2 unten) die Eintragungen der ersten beiden Kästchen (wieder ohne Skalierung und Offset), dann arbeitet Blender mit diesen Formeln:

$X_T = Y_O$; $Y_T = X_O$; $Z_T = Z_O$

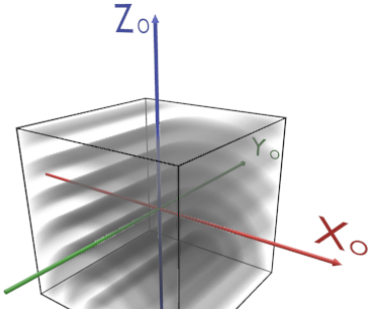
Die Folge: Das Bild wird gedreht, weil X und Y ihre Rollen tauschen.



Dreht man wieder an Size und Offset, so muss man ein wenig aufpassen. Die Werte dort beziehen sich auf die Formeln der Texturkoordinaten. Mit Size X = 3 entstehen die folgenden Formeln:

$X_T = 3 \cdot Y_O$; $Y_T = X_O$; $Z_T = Z_O$

Die Y Koordinate des Objekts wird verwendet, um Positionen in X Richtung der Textur abzulaufen und das im dreifachen Tempo. Das Bild wird in Y Richtung des Objekts gestaucht.



Dich will ich nicht

In jedem Kästchen gibt es noch die Option None. Das bedeutet, dass für die entsprechende Formel einfach gar keine Objektkoordinate verwendet wird. Würde man bei der Bildtextur oben das rechte Kästchen (Z_T) auf None stellen, so würde sich nichts ändern, da in dem Fall Z_O ohnehin keine Rolle spielt.

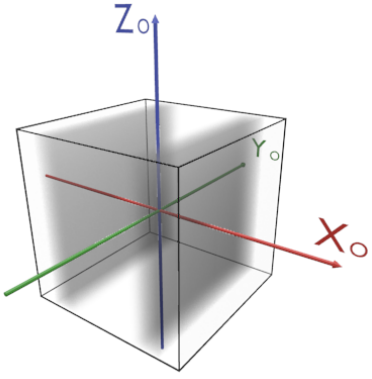
Schalten wir dagegen in der Holztextur von S.4 unten das zweite Kästchen auf None, dann sieht das entstehende Muster überall genauso aus wie in der X-Z-Ebene, es entstehen Röhren. Die Formeln wären nun:

$X_T = 2 \cdot X_O$; $Y_T = 2 \cdot 0 = 0$; $Z_T = 2 \cdot (Z_O + 1)$

Setzt man auch noch das rechte Kästchen auf None, dann ändert sich auch in Z-Richtung nichts mehr und aus den hübschen Kugeln sind öde Streifen geworden. Die Formeln sind:

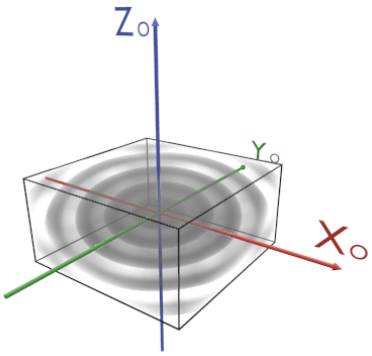
$X_T = 2 \cdot X_O$; $Y_T = 2 \cdot 0 = 0$; $Z_T = 2 \cdot (0 + 1) = 2$

Man sieht in Rechnung und Bild: Der Offset in Z Richtung bleibt erhalten, darum ergibt sich hier kein regelmäßiges Streifenmuster, da nicht die Mittelebene der Textur verwendet wird.[7]



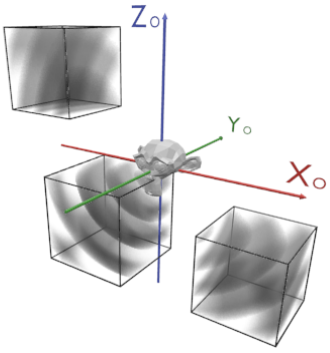
[7] Wer meint das selbe wie oben erreichen zu können, indem man Size Z auf Null setzt, der irrt. Dann wird der Faktor 2 in der dritten Formel zu 0 und damit auch das Ergebnis (unabhängig vom Offset) immer Null. – ätsch!

Kartesische Objektkoordinaten



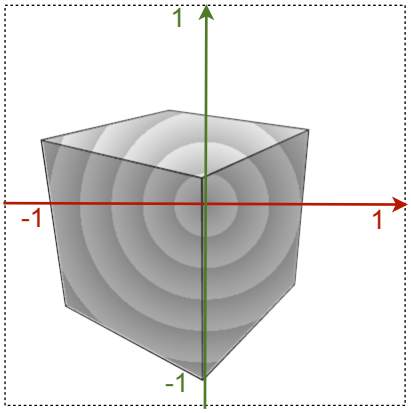
Noch mal Generated

All das Geschriebene ist so weit richtig, Blender besitzt aber noch eine Zusatzoption Namens Map to Bounds, die bei der Skalierung dazwischenpfuschen kann. Ist sie abgeschaltet, dann verwendet Blender tatsächlich, wie oben beschrieben, das Koordinatensystem des Objekts. Wird Map to Bounds dagegen aktiviert, fließen die tatsächlichen Maße des Objekts in die Berechnungen ein. Sind diese nicht gleich groß, so wird die Textur gestaucht.
Vor allem bei animierten Gummibällen ist das wichtig.



Andere kartesische Koordinaten

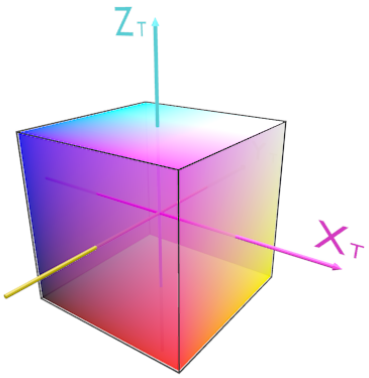
Verwendet eine Textur Generated, so wird sie bei jedem Objekt dessen eigenes Koordinatensystem verwenden. Object dagegen benutzt in allen Fällen das gleiche Koordinatensystem eines einstellbaren Objekts. Im Bild ist Suzanne dieses Objekt und alle Würfel benutzen die selbe Textur.
Global funktioniert genauso, nur dass hier die Weltkoordinaten als Bezug dienen.



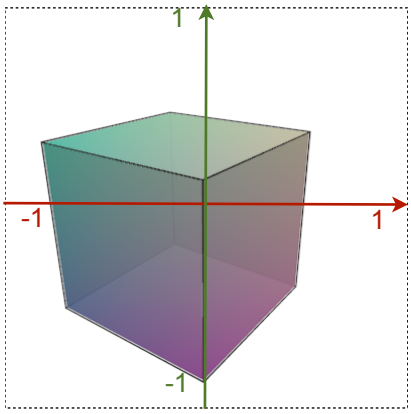
Window

In diesem Fall haben X_O und Y_O mit der 3D Welt nichts zu tun, sondern verlaufen parallel zu den Kanten des Bildes, das berechnet wird. Die Bildmitte ist der Ursprung und die Kanten des Bildes liegen jeweils auf den Koordinaten ±1. Bei einem rechteckigen Bildformat kommt es darum zu Verzerrungen, die man evtl. mit Size ausgleichen muss.
Z_O könnte theoretisch die Tiefe im Bild sein, ist aber in Blender bedeutungslos (Z in eines der Kästchen einzutragen hat die gleiche Wirkung wie None).

Andere Koordinaten



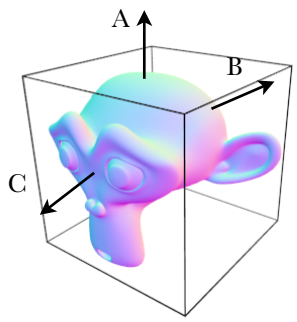
Damit die weiteren Erklärungen und Bilder halbwegs verständlich werden, wird ab jetzt die nebenstehende Textur (Obacht, wir betrachten die hier im Texturraum) verwendet. In ihr steigt der Rotanteil in X Richtung, der Grünanteil in Y Richtung und der Blauanteil in Z Richtung. Dadurch ist sie in der Ecke vorne links unten schwarz und z.B. in der Ecke rechts hinten unten gelb (da viel X, viel Y, aber „kein“ Z).



Erneut Window

Bevor wir Kniffligeres angehen eine Wiederholung: Wenn wir die obige Textur verwenden und mittels Window Koordinaten zuordnen, dann entspricht die horizontale Position eines Bildpunktes X_T, die vertikale Y_T. Z_T wird ignoriert, ist also Null.
Der Effekt ist, dass wir uns aus dem Texturwürfel oben die X_T-Y_T-Ebene in der Mitte herausschneiden und diese über das Bild links spannen. Alle Punkte haben den gleichen mittleren Blauanteil, nach oben wird es „grüner“, nach rechts „roter“.

Normal

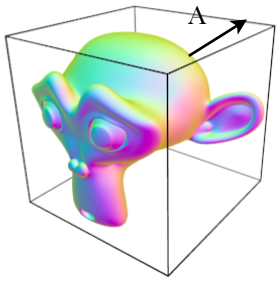


Entscheidend für die Farbe eines Punktes ist, wie der Normalenvektor (die Senkrechte auf der Oberfläche) in diesem Punkt im Vergleich zur Blickrichtung steht. Je mehr er nach rechts zeigt, um so größer ist X_O, je mehr er nach oben zeigt, um so größer wird Y_O und je stärker er auf die Kamera zeigt, um so größer ist Z_O.
Die Pfeile im Bild sollen Normalenvektoren darstellen. A zeigt ausschließlich nach oben, Y_O ist maximal (und damit bei unserer Textur Grün). Der Pfeil ist weder nach links noch nach rechts geneigt, darum ist X_O Null (was bei uns einer mittleren Helligkeit von Rot entspricht). Gleiches gilt für die Z Komponente, denn A zeigt nicht auf uns zu oder von uns weg.
B hat starken Anteil nach rechts und zeigt nach oben, aber nicht auf uns zu oder von uns weg. Die Folge ist viel X_O, etwas weniger Y_O und kein Z_O (viel Rot, ordentlich Grün, Blau auf Mittelwert – Blassgelb). C zeigt auf uns zu, nach links und nach unten, also viel Z_O und X_O bzw. Y_O im negativen Bereich (viel Blau, wenig Rot und Grün).

Stellen Sie sich nun vor, wir drehen Suzanne aus unserer Sicht um 90° im Uhrzeigersinn. A würde nun nach rechts zeigen und Suzannes Scheitel würde in Pastelmagenta eingefärbt. Die Färbung ändert sich also je nach Blickwinkel. Verwendet man beispielsweise nur die Z_O, dann kann man in Kombination mit einem Farbverlauf eine samtartige Oberfläche simulieren oder die Reflexionsstärke einer polierten Oberfläche steuern.

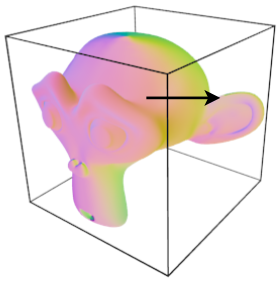
Wer es gern hardcore mathematisch möchte: X_O, Y_O und Z_O sind die Längen der Projektionen auf ein Koordinatensystem aus Blickrichtung (-Z), X Achse der Kamera (X) und der zu diesen beiden senkrechten Achse (Y).

Reflection



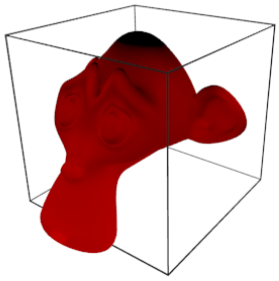
Wir spielen das gleiche Spielchen wie bei Normal, betrachten diesmal aber nicht die Senkrechte auf der Oberfläche, sondern einen gedachten Strahl, wie er aus unserer Sicht an der Oberfläche gespiegelt würde.
Bei A ginge der reflektierte Strahl nach hinten (wenig Z_O) rechts und oben (viel X_O und Y_O). In unserem Fall bedeutet das kaum Blau, viel Grün und Rot und damit insgesamt Gelb.
Speziell mit Bildern als Texturen (am besten Environment Maps) wirkt das Ergebnis tatsächlich wie eine Spiegelung (Spiele tricksen so Chrom hin, ohne echte Spiegelungen berechnen zu müssen).

Tangent



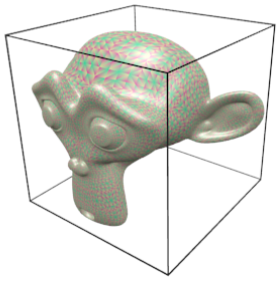
Lassen wir das Objekt in Gedanken um seine Z Achse rotieren und dabei dünnes Garn wie eine Spule aufwickeln (immer direkt auf der Oberfläche, unser Objekt ist extrem klebrig). Die Richtung dieses Fadens ist das, was Blender als sogenannten Tangentenvektor erzeugt, wenn im Material unter Shading Tangent Shading aktiviert ist. Und mit diesem Vektor spielen wir das selbe Spielchen wie schon bei Normal und Reflection.
Über Suzannes Braue würde dieser Faden von links nach rechts verlaufen, parallel zur Bildebene (X_O ist maximal). Also weder vor noch zurück und auch nicht hoch oder runter (Y_O und Z_O sind Null). In unserem Fall also viel Rot plus mittelmäßig Blau und Grün - Rosa!

Stress



Wird ein Objekt z.B. durch Modifier verformt, dann vergleicht Blender die Originallänge von Kanten mit ihrer deformierten Länge. Verlängerung entspricht einem positiven Wert von X_O, Verkürzung einem negativen Wert. Y_O und Z_O sind bedeutungslos.
Da sich elastisches Material wie Gummi oder Haut bei Dehnung verfärbt bzw. glatt gezogen wird und damit mehr spiegelt, finden sich hierfür sicherlich Anwendungsmöglichkeiten. Die genauen Zahlenwerte, die XO hierbei annimmt, konnten nicht ermittelt werden (Tips sind gern gesehen). Vermutlich $X_O = \pm 1$ aber einer unendlichen Dehnung.

UV Koordinaten



Die wurden an anderer Stelle schon intensiv erörtert. Hier also nur der Teil, den das neue Denkmuster mitbringt. Jeder Punkt in jedem Polygon bekommt über UV-Maps die Koordinaten X_O und Y_O im Bereich 0-1 zugeordnet. Wie genau das geschieht, möge man sich in Lektion T4 klar machen. Hat das Objekt noch keine UV-Map, dann wird jedes Polygon auf den gesamten UV-Bereich gespannt. Da Z_O immer Null ist und die anderen beiden Koordinaten nur Werte von 0 bis 1 annehmen, wird Suzanne so mit einem Pastellmix aus Rosa, Hellgrün und Gelb überzogen.

Schlussbemerkung

Das ist keine „Auch das noch!“ Episode, die zu einem bestimmten Ziel führt. Vielleicht ist es aber eine, die die eine oder andere Frage beantworten kann, wenn man wieder einmal nicht versteht, warum eine Textur einfach nicht so will, wie sie soll. Das ist keine leichte Kost, wenn man es nicht schon mal durchgekaut hat, aber das Studium lohnt sich, denke ich.

Und zur Beruhigung all derer, denen das jetzt ein wenig viel war, in Anlehnung an ein mir liebes Zitat Einsteins:

„Wenn Sie meinen, Ihnen würde nach Studium dieses Textes der Kopf brummen,
dann seien Sie versichert:
Meiner brummt nach Schreiben der Sache sicher mehr.“