Auch das noch!

Mathematisches Holz

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

Bildtexturen sind Klasse, wenn man lebendige realistische Oberflächen erhalten möchte. Zumindest natürliche Materialien mit gewissen Regelmäßigkeiten lassen sich aber auch durch Formeln gut nachbilden. Aber wie geht so was?
Gleich als Warnung: Das dürfte mathematisch etwas anspruchsvoller werden. Die grundlegenden Ideen werden aber auch ohne vertiefte Theorie klar. Also hinein ins Vergnügen (ja, solche Basteleien sind für mich genau das)!

Probleme

Platt ist nunmal nicht 3D

Das schönste Bild einer Holzmaserung oder einer Granitoberfläche enthält keine Informationen darüber, wie es einen Zentimeter unter der Oberfläche aussieht. Zersägt man ein Brett, dann verläuft die Maserung auf der Stirnseite vielleicht ringförmig, auf der Oberseite dagegen nahezu parallel. Das funktioniert natürlich auch mit Bildtexturen, wird aber mühsam und wenig flexibel. Ist das Objekt komplexer, dann wird das Auspacken in eine UV-Map kniffliger. Nähte im Verlauf der Textur sind unvermeidlich und beim Beispiel Holz wird das Objekt tendenziell so aussehen, als habe man es in Folie mit Maserung verschweißt (denn genau das ist UV-Mapping). Der Eindruck eines aus einem Holzblock geschnitzten Objekts entsteht nur mit Mühe.

Flexibilität

Man kann hochaufgelöste Bilder verwenden, aber egal wie groß, irgendwann ist man so nah, dass die Kamera einzelne Pixel erkennen kann. Das lässt sich zwar in den meisten Fällen vermeiden, aber in manchen Situationen möchte man auch mal nah ran. Mathematische Texturen kennen keine einzelnen Pixel. Mit etwas Liebe zum Detail verkraften Sie praktisch jeden Kameraabstand mühelos.
Zugegeben, eine prozedurale (durch Funktionen erzeugte) Textur kostet in mehrfacher Hinsicht mehr Zeit. Zum einen ist sie fast nie so schnell erstellt, wie eine Bildtextur (geeignete Fotos vorausgesetzt). Zum anderen kostet sie in bei der Bildberechnung zumeist spürbar mehr Zeit. Ist eine gut durchdachte Textur aber erst einmal fertig kann man sie sehr leicht an individuelle Bedürfnisse anpassen. Der selbe Algorithmus kann Buche, Eiche, glattes Holz oder stark verzerrte Maserung, alt neu oder andere Anpassungen mit dem Verändern einiger Eingabewerte anpassen. Mathematisches Holz kann für völlig neue zwecke „missbraucht“ werden. Prinzipiell kann die in diesem Skript entworfene Holztextur einen Zebrastreifen einfärben, Schichten einer Zwiebel simulieren oder am Bau einer Zielscheibentextur beteiligt sein - genügend Geduld und Kreativität vorausgesetzt.

Lösung

Vorbereitungen

OSL an die Front

Eine Holztextur ist in Cycles auch über Nodes machbar, benötigt aber schon ein wenig Nudelsalat. Aktivieren wir doch Open Shading Language in den Rendereinstellungen unter Render. Dann kann man Shader übersichtlich in einer Textdatei programmieren. Auf OSL selbst werde ich nicht im Detail eingehen, einen Einstieg in dessen Grundlagen findet man z.B. hier: www.openshading.com/osl/getting-started
Zum Arbeiten brauchen wir in Blender einen Texteditor, einen Nodeeditor undzur Ergebniskontrolle einen (kleinen) 3D Bereich. Zusätzlich kommt vermutlich das kleine Textfenster zur Geltung, das beim Start von Blender erscheint (zumindest unter Windows), denn darin werden Fehler des OSL-Skripts angezeigt.
Im Texteditor brauchen wir eine neue Datei, die passend zu benennen ist (wood-pattern.osl wäre nett).

„Rohshader“

Zunächst legen wir die Grundstruktur eines OSL-Shaders mit folgendem Text an:

```
shader wood_pattern
(
    point coord = P,
    point axisbase = point(0,0,0),
    point axistip = point(0,0,10),
    float baseradius = 1,
    float ringwidth = 0.1,
    float ringwidthdelta = 0.1,
    float distortion = 0,

    output float Fac = 0
)

{

}
```

Festlegung der Eingabewerte des Nodes

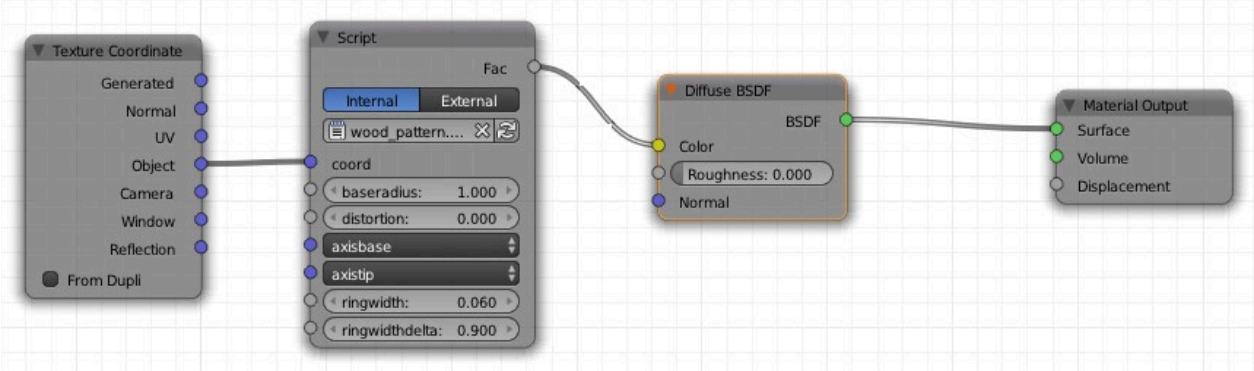
output markiert Ausgabewerte

Das eigentliche Programm kommt zwischen die geschweiften Klammern

Dieser Shader tut noch gar nichts, kann aber schon als Node vom Typ Script eingebaut werden, in dem die neue Textdatei ausgewählt wird. Für später schon mal merken: Nach jeder Änderung am Skript muss man dieses im Script-Node durch Klick auf  neu compilieren. Wenn das nicht klappt findet man im schon erwähnten Textfenster eine Fehlemeldung, die vielleicht (vielleicht auch nicht) weiter hilft.

Grundschaltung

Damit sichtbar wird was man tut genügt eine simple Schaltung mit nur einem Cycles Shader:



Später kann man zwischen Skript und Diffuse noch eine Color Ramp für die passende Färbung des Holzes einbauen, aber während der Konstruktion des Shaders lenkt das eventuell nur ab.

Der mathematische Baumstamm

Eingabewerte

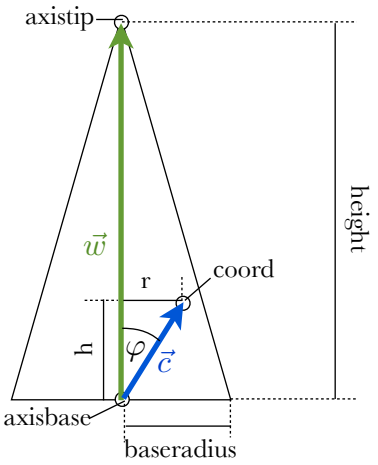
Die folgenden Werte kommen dabei zum Einsatz:

- coord - Koordinaten des Punktes, dessen Farbe zu ermitteln ist.
- axisbase - Position der unteren Stammmitte
- axistip - Spitze des Stamms
- baseradius - gedachter Radius des Stamms auf Bodenhöhe
- ringwidth - mittlere dicke eines Jahresrings
- ringwidthdelta -Anteil, um den Jahresringe von der mittleren Dicke abweichen dürfen
- distortion - als Anschluss für einen externen Noise Node zur Verzerrung der Ringe

Vorberechnungen

Ergänzen wir den Programmteil wie folgt:

```
{
    vector c = coord - axisbase;
    vector w = axistip - axisbase;
    float height = length(w);
}
```



Für die Steuerung der Jahresringe muss klar sein, welchen Abstand r der Punkt coord von der Baumachse hat und in welcher Höhe h des Baums er sich befindet. Da der virtuelle Stamm schräg stehen kann sind r und h nicht x- und y-Koordinate und müssen berechnet werden. Dafür ist ein wenig analytische Geometrie unvermeidbar (bei Bedarf: Nachschlagen).
Für das Skalaprodukt gilt

$$\vec{w} \circ \vec{c} = w \cdot c \cdot \cos\varphi$$

Verwendet man statt $\vec{w}$ auf den zugehörigen normierten Vektor $\vec{w}_0$ der Länge 1 wird daraus

$$\vec{w}_0 \circ \vec{c} = 1 \cdot c \cdot \cos\varphi = c \cdot \cos\varphi$$

Das ist aber nichts anderes als die gesuchte Länge h. Wir ergänzen das Skript passend:

```
float h = dot(normalize(w), c);r erhält man mittels Pythagoras:
```

Der Radius r folgt mittels Pythagoras:

$$r = \sqrt{c^2 - h^2}$$

Im Skript:

```
float r = sqrt( pow(length(c), 2) - pow(h, 2));
```

Für Holz braucht man eine Säge(zahnfunktion)

Die Färbung des Holzes soll in Abhängigkeit von r die verschiedenen Färbungen eines Jahresrings durchlaufen. Die Ringposition rp berechnen wir dafür zunächst extrem simpel:

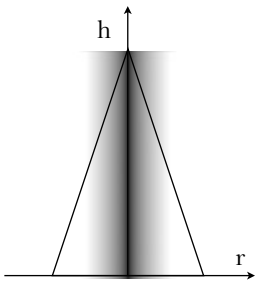
```
float rp = r;
```

Diese wird dann noch an die Ausgabe übergeben:

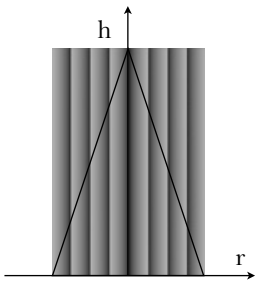
```
Fac = rp;
```

Direkt auf der Baumachse hat die Textur jetzt den Wert 0, der von dort mit dem Abstand ansteigt. Aber Werte oberhalb von 1 machen für eine Textur keinen Sinn, von einer Wiederholung ist auch noch keine Spur und damit stimmt auch die Ringbreite nicht. Das erreichen wir in einem Aufwasch, indem wir den Abstand durch die Ringbreite teilen und vom Ergebnis nur die Nachkommastellen nutzen. Das geht z.B. so

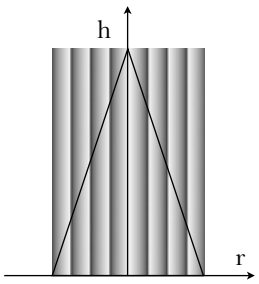
```
float rppart = rp / ringwidth;
Fac = rppart - floor(rppart);
```



Nur r steuert den Wert



Mit der Funktion mod()



Korrigiert wie unten angegeben.

Damit das Muster in der richtigen Richtung durchlaufen wird (der Teil mit den Werten um 0 soll das sog. Spätholz, der schmale dunkle Teil sein, nach dem die Färbung abrupt zum helleren Ton umbricht) verwenden wir -rp statt rp. Die endgültige Ausgabe lautet:

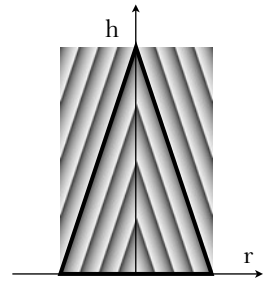
```
float rppart = -rp / ringwidth;
Fac = mod(-rp, ringwidth);
```

Alle weiteren Eingriffe erfolgen bei der Berechnung von rp. Zunächst sorgen wir dafür, dass die Linien nach oben hin zur Achse verlaufen:

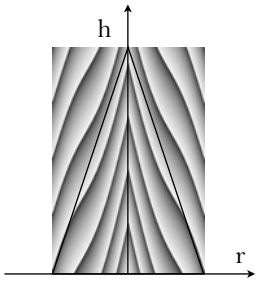
```
float rp = r
           + baseradius * h/height;
```
Semikolon neu setzen!

Die Verschiebung ist so angelegt, dass der Jahresring, der am Boden den Radius baseradius besitzt in der Höhe height genau durch die Spitze verläuft.
Das Muster ist noch zu regelmäßig. Mit noise(„perlin“, r + h*0.04) wird ein Zufallswert in Abhängigkeit vom Radius (und ein klein wenig in Abhängigkeit von der Höhe) ermittelt. Diesen Multiplizieren wir mir ringwidthdelta und addieren ihn einfach zu rp dazu. Und da wir schon dabei sind ergänzen wir auch noch durch die Verzerrung.

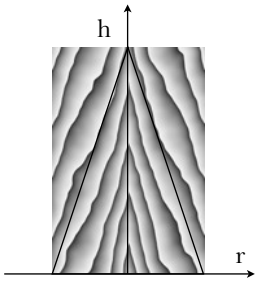
```
float rp = r
           + baseradius * h/height
           + noise("perlin",r+h*0.04) * ringwidthdelta
           + distortion;
```
Semikolon neu setzen!



Schräg verlaufende Streifen



mit unterschiedlicher Dicke

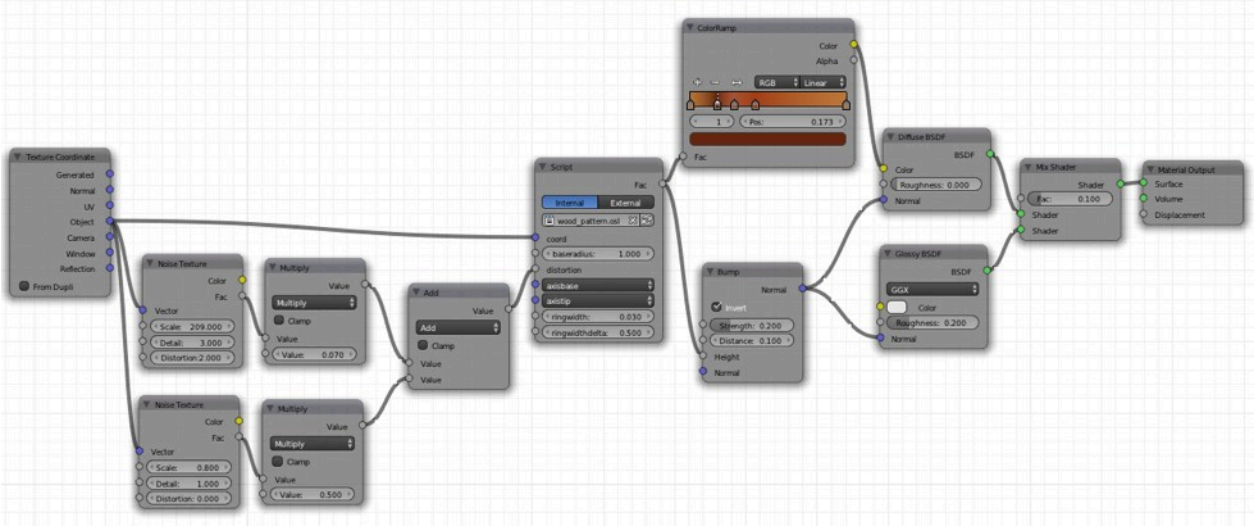


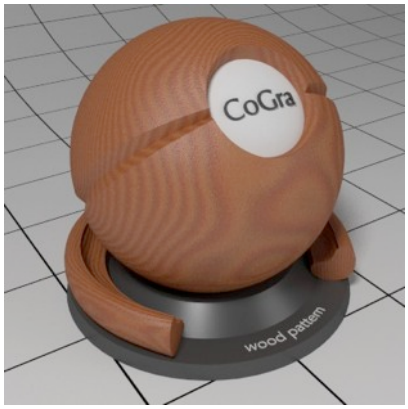
und ein wenig Verzerrung

Man könnte die Steuerung noch weiter verfeinern, indem man die 0,04 durch einen weiteren Eingabewert ersetzt, so dass man auch den Einfluss der Höhe auf die Streifenbreite noch gezielt steuern kann. Zunächst soll es aber so genügen.

Resultat

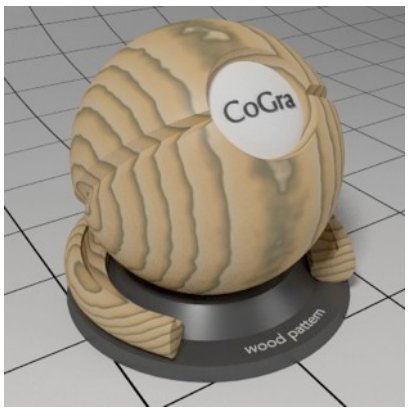
Das so erstellte Holzmuster ist ein Hilfsmittel, keine Komplettlösung. Indem man die Schaltung vom Anfang noch ein wenig erweitert ergibt sich aber schon ein annehmbares Ergebnis.



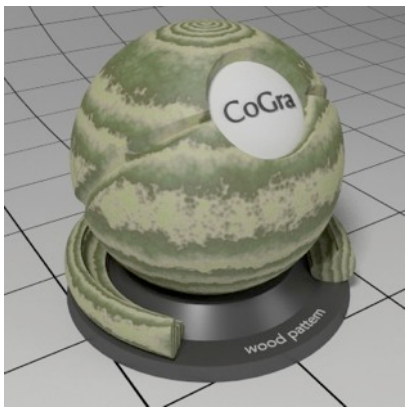


Eine Annäherung an das Foto zu Beginn

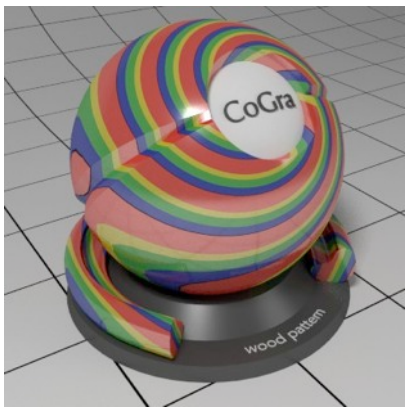
Und noch drei kleine Alternativen:



Vielleicht eine Ahornart



Schnellholz (wenn Sie das kennen)



Farbspiralen