Material-Koeffizienten

Table Of Contents

1 Einleitung

Im Material-File des Typs *.mtl gibt es einen von mir noch nicht näher behandelten Parameter `coeffs`, welcher 4 Werte aufweist:

Code: myMaterial.mtl

```
1. ...
2. props = {
3. coeffs = {
4. 1, 1, 0.25, 20,
5. },
6. },
7. ...
```

Wofür stehen diese 4 Werte, was beeinflussen sie? Das möchte ich im Folgenden einmal versuchen zu klären. Ich habe mir hierzu ein simples Asset gebaut, einfach einen Würfel mit einer 256x256px Textur, die auf allen Seiten gemappt wurde. Dieser Würfel wird 4mal dargestellt, mit bis zu 4 unterschiedlichen Textur-Mustern. Einmal ein Schwarz-Weiß Streifenmuster, danach das selbe nur mit einem Verlauf zwischen den Farben, gefolgt von einem rot-grünen Wolken-Muster und schlussendlich noch eine Metalltextur, wie sie oft bei Trittstufen oder dergleichen Anwendung findet. Hier nun einmal dieser Testaufbau:

Wenig spektakulär, aber hoffentlich nützlich. Für die erste Testreihe, habe ich jeden Paramter einzeln geändert bei der gleichen verwendeten Textur, damit die Unterschiede besser Sichtbar werden. Ich benenne die Würfel von links nach rechts mit 1..4. Somit also zu den Ergebnissen für

2 Parameter 1

Würfel 1 bekam die oben gezeigten, im Folgenden als Standard bezeichneten, Werte, also `1, 1, 0.25, 20`. Würfel 2 bekam die Werte `2, 1, 0.25, 20`, Würfel 3 `4, 1, 0.25, 20`, und Würfel 4 `8, 1, 0.25, 20`, also das doppelte an. Welcher Effekt zeigt sich nun im

Da pures Weiß in TF ein bischen heikel ist, sobald man HDR aktiviert hat, habe ich es nochmals mit der Farbtextur des 3. Würfels probiert, hier erkennt man es etwas besser. Man kann schön erkennen, wie man einen Leuchteffekt erzeugt, wie man ihn bspw. für Fahrzeuglampen benutzen kann. Wie kann man diesen Effekt in seiner Wirkung beschreiben? Nun, er hellt die Textur auf und minimiert zunehmend die Auswirkung des Shadings, also der Schattierung der Sonnenabgewandten Flächen. Genau das wirkt dann so super bei Fahrzeug Scheinwerfern zum Beispiel. Auch wenn sie in einer schattierten Fassung oder eben nur auf der Sonnenabgewanten Seite sitzen, sie leuchten dennoch schön. Diesen Effekt kann man sehr schön bei meinem Flirt3 sehen, um mal ein Anwendungsbeispiel zu geben:

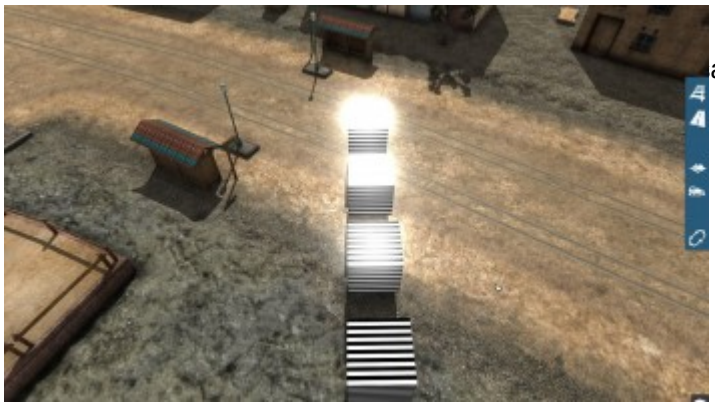
Die Scheinwerfer liegen im schattierten Bereich und in dieser Einbuchtung drin und leuchten dennoch schön. Wie man an den Test-Würfeln aber auch erkennt, sollte man die Werte nicht allzu hoch drehen. Meistens wirkt ein Wert im Bereich von 3 oder 4 am Besten, 5 ist oft schon zu hoch. Aber hier gilt: Probieren. Je dunkler die Ausgangsfarbe, desto höher muss der Wert gesetzt werden. Mein rotes Rücklicht musste ich bspw stärker "beleuchten" wie die weißen Scheinwerfer.

3 Parameter 2

Gut, was bewirkt nun der zweite Parameter? Hier habe ich wieder die selbe Steigerung der Werte genutzt, `1, 2, 0.25, 20,, 1, 4, 0.25, 20,` und

Auch hier erfolgt eine Leuchtwirkung, allerdings nur auf den Sonnen zugewandten Seiten. Das Shading wirkt wie gehabt. Mangels Kreativität fällt mir spontan kein Anwendungsbeispiel ein 😕 Falls ihr Ideen habt, nutzt die Kommentare und ich werde es anfügen 😆

4 Parameter 3



genutzt: `1, 1, 0.25, 20,, 1, 1, 0.5, 20,` ah nun so aus:

Bei Bild1 lässt es sich eher erahnen, bei Bild2 wird es aber doch recht deutlich: Hier wird der Reflektionsgrad angegeben. Daher musste ich auch die Kamera etwas drehen, damit der Sonnenlicht-Spiegel-Effekt in Kraft tritt. Je höher dieser Wert also ist, desto stärker tritt dieser Glanzeffekt auf. Matte Materialen sollten also eher einen geringen Wert oder gar 0 haben, während glänzende einen höheren vertragen können. Beispielsweise braucht Holz hier keine großen Werte, lackiertes Holz kann aber durchaus etwas glänzen. Im Übrigen sei hier dringend auf den Unterschied zwischen Glanz und Reflektion hingewiesen! Aber darauf werde ich später noch etwas tiefer eingehen. Kommen wir also zu

5 Parameter 4

Zu diesem Parameter ein paar Hinweise: Der dritte Parameter sollte nicht zu niedrig gewählt sein, da der vierte szs. mit diesem zusammen hängt. Desweiteren sollte die Textur nicht zu hell sein, da sonst der Überstrahl-Effekt (Bloom) wieder alles zunichte macht. Und ganz wichtig: Ihr braucht ein Objekt mit

gerundeten Flächen! Ich nutze also wieder die rot/grüne Wolkentextur auf eine Reihe von Kugln gepappt gepaart mit den folgenden Werten: `1, 1, 0.75, 0.5,`, `1, 1, 0.75, 1,`, `1, 1, 0.75, 5,` und `1, 1, 0.75, 20,`. Wieso man was rundes braucht, offenbart sich im folgenden Bild:

Dieser Parameter beeinflusst also die Stärke des "Reflexpunktes". Je höher der Wert, desto präziser ist der reflektierte Spot auf der Oberfläche. Polierte Flächen sollten hier also eher einen hohen Wert haben und rauhe einen eher niedrigen.

Soweit zu den Koeffizienten der Materialien. Es lassen sich aber noch andere Tricks bewerkstelligen. Dazu benötigen wir aber einen Ausflug in die Texturen. Fangen wir also an mit dem...

6 Alphakanal der Textur

Was ist dieser Alphakanal? Ein Bild auf dem Monitor wird aus den 3 Farbbausteinen Rot, Grün und Blau zusammen gesetzt. Diese 3 Farben werden in 3 "Kanälen" gespeichert. Jedes Pixel wird dabei mit 8 Bit pro Kanal gespeichert. Also 8 Bit für den Rotanteil des Bildes, weitere 8 Bit für den Grünanteil und nochmal 8 Bit für den Blauanteil. Daraus ergibt sich am Ende eine für uns interpretierbare Farbe. Ein Pixel benötigt also 24 Bit Speicherplatz. Natürlich kann man auch 16 Bit je Farbkanal speichern, was feinere Farbverläufe zur Folge hätte, "üblich" sind aber 8 Bit. Diese 8 Bit als dezimaler Wertebereich ausgedrückt sind die Berühmten 256 Farbnuancen, also Rot kann Werte von 0..255 annehmen, genauso Grün und Blau.

Will man nun aber, dass zwei Farben sich überlagern, so muss man einen gewissen Faktor angeben, eine Art Gewichtung, zu wievielen Teilen jede Farbe in die endgültige Farbe einfließt. Beispielsweise haben wir eine pure rote Farbe und eine pure Grüne. Nun soll die Rote zu 75% die Grüne überlagern. Man sagt auch, man blendet die Farben übereinander. Dabei gilt die Formel `resultierende Farbe = alpha * Farbe1 + (1- alpha * Farbe2)`. Das bedeudet, dass man in unserem Beispiel 75% von der roten Farbe mit 25% der Grünen mischt (und man hat am Ende wieder 100% Farbe -> 75+25). Da dieser Faktor eben als alpha bezeichnet wurde, hat sich dies als begriff festgesetzt und diese Technik wird auch Alpha-Blending genannt. Nun, was hat das mit dem Alpha-Kanal zu tun?

Wie die anderen drei Kanäle, können Texturen einen zusätzlichen vierten Kanal besitzen. Bei 8 Bit Kodierung kann dieser pro Pixel wiederrum Werte von 0..255 annehmen und pro Pixel werden nun 32 Bit Speicherplatz benötigt. Und dieser Kanal speichert nun keine weitere Farbe (etwa Gelb), sondern diesen Gewichtungsfaktor, welcher eben alpha genannt wurde. Daher hat der vierte Kanal auch seinen Namen: Alphakanal. Klassisch kann man damit also pixelgenau die Transparenz der Textur angeben. Sozusagen eine Transparenz- oder Alpha-Maske. TF tut dies ebenfalls, wenn man den Material-Typ `map_color_alpha` verwendet. Nutzen wir aber den Typ `map_color_reflect`, so wird dieser Kanal anders interpretiert. Wie in meinem Lexikon-Artikel zu den Materialien bereits erläutert, gibt man hier zusätzlich eine Environment-Map an. Eine Cube-Texture - also 6 quadratische Texturen, die zusammengesetzt die Innenwände eines uns umgebenden Würfels ergeben - welche unsere Umgebung darstellen soll. Normalerweise müsste man für einen realistischen Effekt diese Cube-Map dynamisch erstellen, da dies aber jede Menge Performance kostet, wird gerne solch ein Umweg genommen. Diese Textur soll die Umgebung darstellen, die bspw .von Fenstern reflektiert/gespiegelt wird.

Unser Alphakanal gibt nun also nicht mehr an, wie transparent die Textur sein soll, sondern wie spiegelnd.

Würfel 1 und 4 sind einfach ohne Spieschele :) der zweite hat das Muster von Würfel 1 oben als Alpha-"Muster" (also Balken mit voller Transparenz gefolgt von Balken völlige Deckkraft) und der dritte das von Würfel 2 (diese "Balken" gehen fließend ineinander über). Wie man sieht spiegelt sich oben drauf der blaue Himmel (also der unserer Umgebungs-Textur) im entsprechenden Muster und auch an den Seiten ist der Effekt zu erkennen.

Als nächstes habe ich einmal die Textur mit voller Deckkraft, mit 33% Transparenz, 66% Transparenz und voller Transparenz geladen:

Man sieht schön, wie der Effekt zunimmt. Man kann die Transparenz ganz gezielt mit einer Alpha-Map steuern. Dazu erst einmal ein paar Begriffserklärungen, die wir auch später noch brauchen werden:
Eine Color-Map oder auch Diffuse-Map genannt, ist die eigentliche Textur, welche die farbe auf unser Objekt zaubert.
Die Alpha-Map ist eine schwarz-weiß Textur, welche angibt, welche Bereiche mehr oder weniger transparent sein sollen.
Eine Bump-Map ist ebenfalls eine schwarz-weiß Textur, diese gibt jedoch Höhenunterschiede auf der Textur an.

Eine Normal-Map kann aus einer Bump-Map generiert werden und beeinflusst die Flächen-Normalen, was das Shading beeinflusst.

Was macht nun also eine Alpha-Map? Je dunkler ein Pixel auf der Alpha-Map ist, desto transparenter wird der Pixel der Color-Map. Man erstellt eine Alpha-Map also, indem man zum Beispiel einfach seine Fenster der Textur markiert und in einer neuen Ebene diese sehr dunkel wieder einfärbt. Das ganze über eine Weise Hintergrund-Ebene gelegt und fertig ist eine Simple Alpha-Map. Oben habt ihr sowas schon gesehn, als ich das Streifenmuster des ersten Würfels als Alpha-Map für die Metall-Textur genutzt hatte. Nur die Bereiche, die schwarz waren, haben am Ende auch die Umgebung reflektiert. Um eine Alpha-Map anzuwenden, nutzt zum Beispiel GIMP, ladet die Alpha-Map, kopiert sie (alles markieren und kopieren) und ladet dann eure eigentliche Textur. Rechtsklickt auf die Ebene und fügt eine Ebenenmaske hinzu. Hier kopiert ihr nun eure Alpha-Map hinein und speichert das Ganze wieder als tga (bei GIMP "exportieren").

Dies war aber nur der billigste Trick. Viel Effekt könnt ihr mit dem nächsten Schritt erreichen: Einer...

7 Normal-Map

Dieses gute Stück löst bei vielen (verständlicherweise) Kopfzerbrechen aus. Was ist eine Normal-Map. Es ist etwas ziemlich mathematisches und man muss ein wenig wissen, wie Shader funktionieren. Also Alltagszeugs Aber fangen wir mal beim Urschleim an: Wieso heisst das Ding eigentlich Normal-Map? Map ist eventuell schon klar, damit wird allgemeinhin eine Textur bezeichnet. Die Textur gibt wie eine Karte an, wo was ist. Hier ist was rotes, da was blaues... Jeder Pixel hat seine Farbe usw usf. Gut, auch eine Normal-Map ist prinzipiell erstmal nichts weiter, wie eine Textur. Der Ein oder Andere kennt auch, wie die in etwa aussehen: Und nein, bitte sagt jetzt NICHT Blau! Das ist kein Blau *argh Das ist so ein Lila-Ton. Aber gut, ich wollte ja erklären, wieso das Ding Normal-Map heist.

Eine Normale ein ein Begriff aus der Vektoren-Lehre. Ein Vektor ist ein Punkt oder eine Richtung. Im 3 dimensionalen Raum schreibt man einen Punkt als P(x|y|z|0) und eine Richtung als R(x|y|z|1). Warum? Der vierte Wert trägt den Bezeichner w. Dieser ist ein Skalierungs-Faktor für den Vektor. Ein Punkt ist ein fest definierter... nuja, Punkt eben, im Raum. Unveränderlich szs. Skaliert man einen Punkt, so erhält man nur wieder den Punkt, da w = 0 ist. Anders bei Richtungen. Diese kann man beliebig skalieren und ihre Länge beeinflussen. Eine Richtung reicht vom Ursprung bis zur angegebenen Koordinate. Verlängert man diese Richtung, ändert sich zwar die Koordinate (da w = 1), die Richtung selbst bleibt aber die gleiche. Richtungen kann man an jeden beliebigen Punkt quasi anheften. Man kann sie also verschieben und skalieren, die Richtung bleibt die gleiche. Nur drehen kann man sie natürlich nicht, ohne die Richtung selbst zu ändern.

So, eine Richtung, unendlich viele Versionen? Das is doch doof! Wie kann ich eine Richtung am besten "einheitlich" darstellen? Tjo, wo wirs schon erwähnen, mit einem Einheitsvektor. Was ist ein Einheitsvektor nun? Das ist ein Vektor mit der festen Länge 1! Man eliminiert also die Länge der Richtung, indem man sie standartisiert - man sagt auch... *Trommelwirbel* normiert! Ein solcher Einheitsvektor wird auch als Normalen-Vektor - oder kurz Normale - genannt. Und das ist auch schon des Rätsels Lösung: Eine Normal-Map speichert keine Farben, sondern normalen-Vektoren - normierte Richtungen! Wie geht das? Nunja, im 3 dimensionalen Raum haben wir eben 3 Dimensionen: x, y und z. Und unsere Farben setzen sich aus 3... ah! R, G und B! Im roten Kanal werden also die x-Werte gespeichert, im grünen die y-Werte und im blauen die z-Werte. Dabei sei nun aber folgendes zu bedenken:

Ein normierter Vektor wird auf einer Achse *niemals* über den Wert 1 hinauskommen. Der Wertebereich für unsere Koordinaten liegt also bei -1..1, der unserer Farben bei 0..255. Das eine wird nun aufs andere abgebildet. Hat R den Wert 255, so bedeuted das für x, dass es den Wert 1 hat. 191 resultiert in 0.5 usw usf. Grafikengines lesen Texturwerte sowieso nicht als 0..255 aus, sondern direkt als 0..1, von daher ist das eh Wurscht. Aber nützlich zu wissen. So, wir wissen also, wieso das Ding heisst wie es heisst und was es speichert und wie es das tut. Aber wieso zum Henker ist das immer so lila?

Das kommt daher, dass eine Fläche, also ein Dreieck (eine Ebene wird eindeutig von 3 Punkten definiert, ein weiterer kann auf der Ebene liegen, muss aber nicht) generell als der x/y-Ebene zugehörig angesehen wird. Stellt euch ein Koordinatensystem vor das vor euch liegt. x zeigt von euch fort, y nach rechts und z zeigt nach oben zur Decke. Nehmt ihr nun eine beliebige Fläche her, so wird sie im Sinne der Normal-Map platt auf den Tisch gelegt und weicht nicht in z-Richtung ab. Dabei ist es egal, wie die Fläche im eigentlichen Raum liegt. Hier noch auf die Feinheiten der verschiedenen Bezugssysteme (Model-Space, Camera-Space, World-Space...) einzugehen, erspare ich mi Die Flächennormale zeigt immer im 90° Winkel von der Fläche weg, also in unserem Fall nach oben zur Decke. Es hat also den Vektor 0|0|1. Im RGB-System wäre das also? 0|0|255? Falsch Normalen können sich rundherum im Raum drehen, also auch bis -1. die 0 liegt also in der Mitte unseres RGB-Raumes, also sind die korrekten Werte 127|127|255 - und ratet mal, was das für eine Farbe ergibt

So, wie beeinflusst nun eine Normal-Map die Textur? Dazu muss man wissen, was beim Shading passiert. Wie schon angesprochen, bedeuted shading nichts weiter wie schattieren. Dazu wird die Flächennormale mit dem Richtungsvektor des Lichts verglichen. Je nach Winkel ist die Fläche dann voll beleuchtet, leicht oder doll schattiert oder liegt einfach (wenn sie vom Licht weg zeigt) direkt im Schatten (dann zählt nur noch das Ambient-Light). Um diese Helligkeit zu simulieren, wird die Farbe einfach multipliziert. Halbe Helligkeit: Farbe * 0.5 und fertig. Voller Schatten wäre demnach *0 was wiederrum in 0 Resultieren würde und das bedeuted volles Schwarz. Raben schwarzen Schatten haben wir aber nicht, das kommt durch besagtes ambient-light, was ebenfalls hinzugerechnet wird, um das Ganze wieder aufzuhellen. Nur so als Randinfo. Wir nehmen also mit: Schattierungen errechnen sich aus dem Winkel der Flächennormale mit der Lichtrichtung. Und was beeinflusst unsere Normal-Map? Die Flächennormale. Sie liefert ein pixelgenaues Muster, welcher Teil der Fläche in welche Richtung schauen soll, obwohl die Fläche doch eigentlich flach wie eine Flunder ist. Und damit erzeugt an Schattierungen und damit den Eindruck von Oberflächenstrukturen, wo garkeine sind.

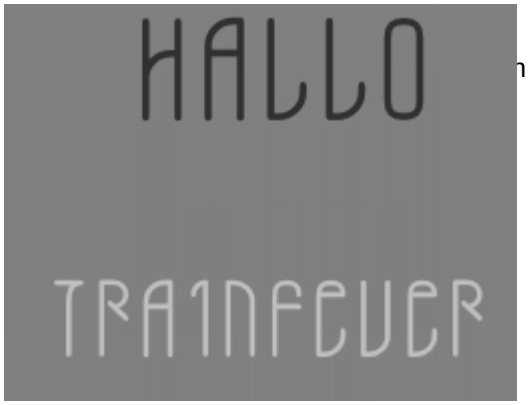
Wie erzeugt man denn nun am dümmsten so ein Ding? Mit dem Taschenrechner jeden Pixel einzeln berechnen? Ginge sicher, aber äääh Die wohl einfachste Methode wäre eine...

8 Bump-Map

Wie oben angedeudet, ist auch dies eine schwarz-weiß Textur. Man sollte sich eine Grundfarbe aussuchen, von der aus man weiter arbeitet. Ich nehme gerne ein 50%iges Grau, dann hab ich Luft nach "oben" und "unten". Eine Bump-Map wie folgt interpretiert: Es werden benachbarte Pixel verglichen, also deren Grau-Wert. Je dunkler die Farbe, desto tiefer liegt der Pixel, je heller, desto höher ist er. Liegen nun 2 Pixel "gleich hell" oder "gleich dunkel" nebeneinander, so erfolgt kein Höhenunterschied und diese Stelle bleibt flach. Ändert sich die "Höhe", wird dies als Steigung oder Gefälle interpretiert. Ich erstelle also wie gesagt erstmal ein 50% Grau und male all die Erhöhungen (zum Beispiel Gummiwülste) heller nach und alle Vertiefungen (Lüfterschlitze oder Karosserie-Spalten als Beispiel, aber auch Türgriffe) eben dunkler. Manche schwören auch auf das umwandeln des Original-Bildes/Textur in ein Schwarz-Weis Bild um das als Bump-Map zu nutzen, aber der Effekt kommt dem Gewünschten höchstens nahe. Es ist ja richtig, dass durch die Höhenunterschiede eines fotografierten Irgendwas hellere und dunklere Bereiche in Bezug zu den wirklichen Höhenverhältnissen zu finden sind, jedoch werden Schatten ja "geworfen", ergo hat man die Dellen dann

meist an den falschen Stellen. Hinzu kommt, dass ein flacher Fleckiger Stein bspw durch durch die reine Farbgebung zu einer Bump-Map führt, die mit dem Original nichts mehr zu tun hat. Empfinde ich als ungeeignete Lösung. Daher male ich mir die Dinger lieber selber.

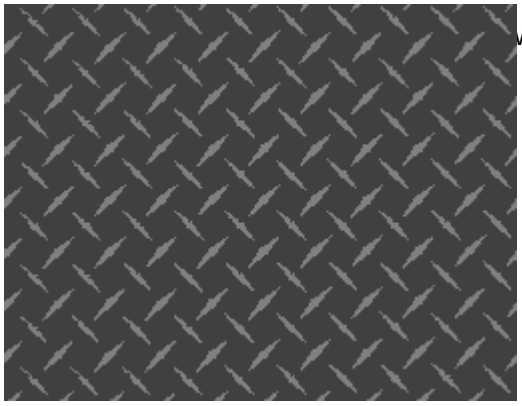
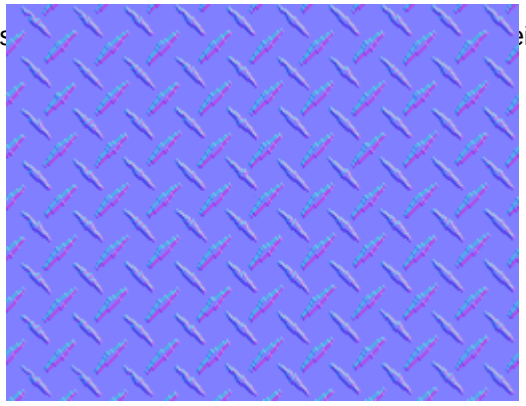
Nun gibt es diverse Möglichkeiten, aus solch einem Graustufen-Bild eine Normal-Map zu generieren. Zum einen gibt es diverse Plugins für GIMP oder Photoshop bspw, zum anderen gibt es auch StandAlone Tools. Ich nutze begeistert xNormals. Ihr ladet eine Bump-Map und generiert die Normal-Map. Eine Besonderheit gibt es: Speichert ihr die Normal-Map, so werden da Transparenzen mit rein gemehrt. Das finde ich persönlich äusserst unpraktisch, da TF diese auch auf seine Weise interpretiert, und ich das doch gerne wieder selber Steuern möchte. Ihr könnt sie aber auch kopieren, dann könnt ihr sie eurer Projekt-Datei (also die, wo eure ganzen Textur-Ebenen mitgespeichert werden -> pdn oder psd oder xcf...) hinzufügen und habt sie ohne Transparenzen. Da ich meine eh immer als seperate Ebene mit abspeicher, find ich das also sehr praktisch so.



Gut, damit haben wir alles durch? Falsch! Ein Thema, was auch für mich neu ist:

9 Transparenzen in Normal-Maps und Beeinflussung durch coeffs

Ich hatte das schonmal bei meinem O309 Bus von Mercedes probiert. Am Dach. Es zeigte sich keine Wirkung und ich gab entnervt auf Mittlerweile glaube ich zu wissen, woran es lag: An diesen elenden hellen Farben. Da kommt bei TF einfach nix durch *grml* Also starte ich jetzt einen neuen Feldversuch. Was sollen Transparenzen in Normalmaps bringen? Ich sprach ja schon an, dass TF diese ebenfalls interpretiert. Aber wie? Ich hab meinen vierten Würfel diese Eisen-Textur da gegeben. Das ist so ein Metall/Stahl Dingens, wo viel drauf rumgelaufen wird, sprich diese "Striche", die sich so für Griffigkeit beim Laufen wie Noppen erheben. Daher sind die quasi Blank poliert, Eventueller Lack/Rostschutz ist abgetragen und das Blanke Metall glänzt darunter durch. Die anderen Bereiche der Textur glänzen allerdings nicht, sind eher matt. Was wir bisher kennen gelernt haben, kann so einen Effekt nicht umsetzen. Entweder es glänzt alles, oder nichts. Glücklicherweise ist diese Textur gut geeignet um sehr simpel eine Normal-Map zu erzeugen. Also auf dem Weg, den ich eigentlich nicht leiden kann

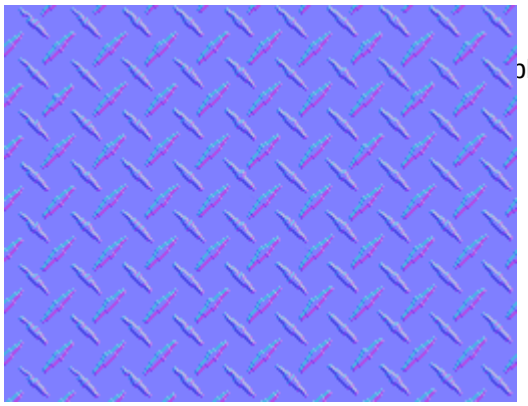



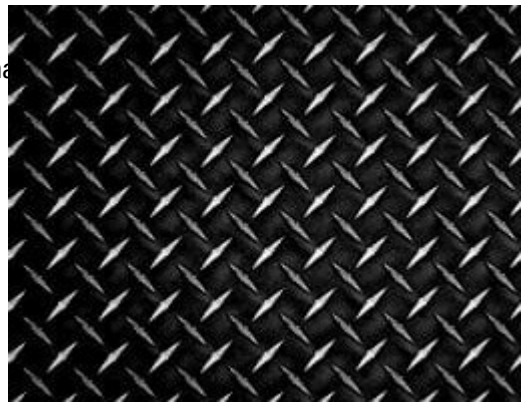
Gerade an der Seite sieht man sehr schön, wie der Effekt der Normal-Map immer deutlicher heraustritt. Leider geht eben dieser Leuchteffekt damit einher. Warscheinlich muss man gleichzeitig den ersten Parameter absenken, um dem entgegen zu wirken. Wobei das ja auch die Schattierten Flächen beträfe *hmm*

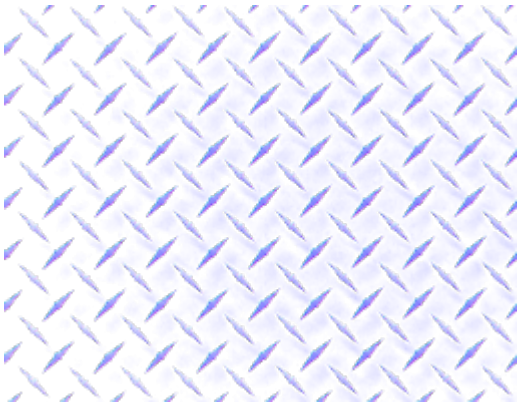
Wie man sieht, ist es mir nicht gelungen, den Effekt gescheit auszugleichen. Bei den ersten Dreien scheint es ganz gut zu passen, wenn man von der Fläche oben drauf ausgeht, dafür passen die Seiten aber überhaupt nicht. Und hinten der leuchtet wie ein Weihnachtsbaum oben drauf. Sofern hier keiner andere

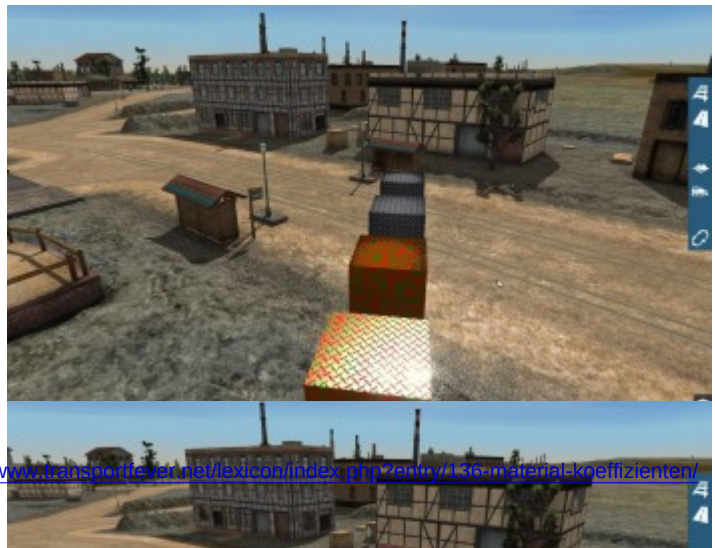
Ideen der Anwendung hat, würde ich empfehlen, den einfach auf 1 zu lassen

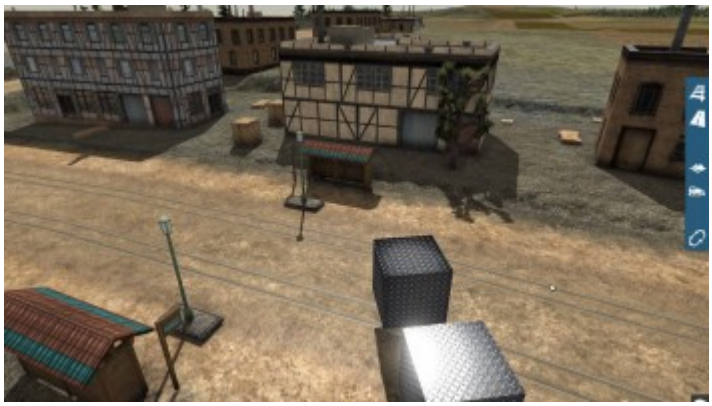
Ok, wollen wir mal weiter machen im Text. Soweit ich weis, wird nämlich der dritte Wert der coeffs ebenfalls mit der Normal-Map verrechnet (multipliziert). Und zwar die schon angesprochene Transparenz dieser. Wenn wir also nur das abgewetzte Metall schön glänzig wollen und den Rest nicht, dann müssen wir unserem Material also einen hohen Wert geben. Ich sag mal Testweise 2. Dieser wird mit der Transparenz der Normal-Map multipliziert. Schwarz ist 0 und 4*0 ergibt wieder 0 und würde also den Effekt deaktivieren. Je heller, desto mehr vom eigentlichen Effekt kommt also durch - soweit zur Theorie. Erstellen wir uns also eine Alpha-Map, die ähnlich der Bump-Map aussieht.



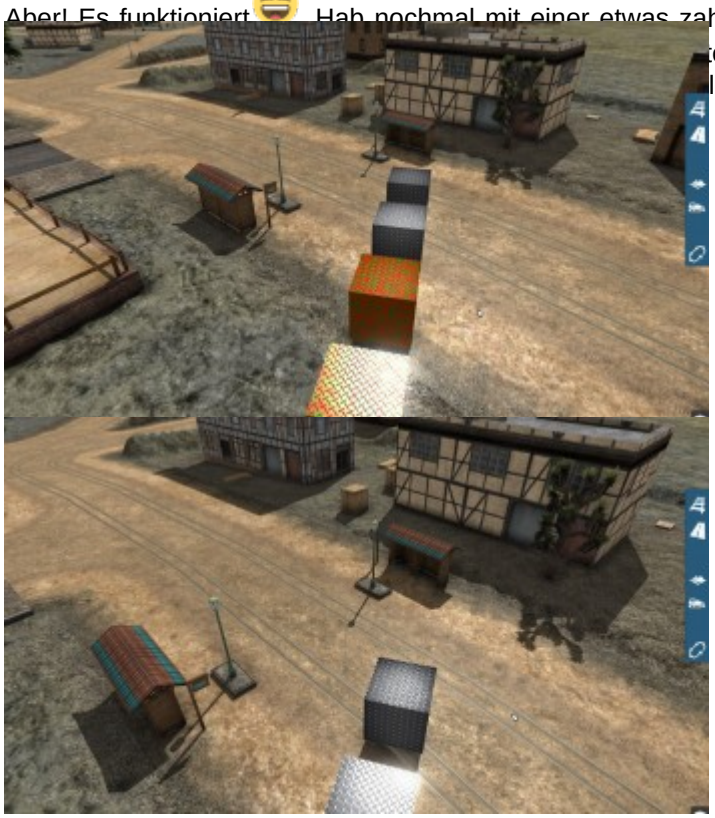


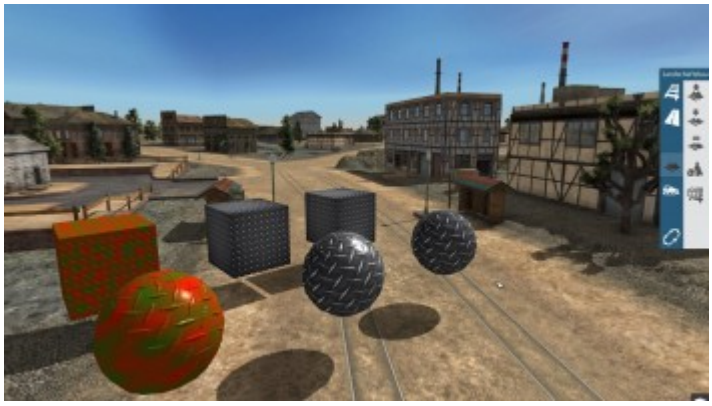






Der Effekt ist definitiv erkennbar, aber noch nicht wirklich schön. Ich vermute, dass man auf komplette Transparenz besser verzichtet und nur maximal ein dunkles oder gar nur mittleres Grau verwenden sollte. Aber! Es funktioniert 😆 Hab nochmal mit einer etwas zahmeren Alpha-Map rumgespielt, aber ich furchte, [illegible]er Spot entstehen kann. So wirklich zufrieden bin [illegible]lder:





Dann hoffe ich, dass euch dieser Artikel etwas geholfen hat und freue mich auf viele schöne Materialien in Zukunft 😛