Materialien

Inhaltsverzeichnis

In TrainFever gibt es verschiedene Dateiformate, welche unterschiedlichen Zwecken dienen. Hier soll es nun um die .mtl - die Materials - gehen.

1 Struktur einer .mtl-Datei

Der Grundaufbau einer .mtl-Datei sieht wie folgt aus:
Code

```
function                                                                          data()
return                                                                                 {






        },


        [...]






            },

        },



        },

    },


}
end
```

Alles anzeigen

1.1 Struktur der Material-Typen-Angabe

Hier habe ich mittig etwas ausgeschnitten [...], denn das soll der Kern dieses Eintrages werden: Die unterschiedlichen Möglichkeiten der "Einstellungen". Meinem jetzigen Verständnis nach können das bis zu 3 Blöcke sein, die jeweils wie folgt aussehen:
Code

```
<material_type>                              =                              {



        },
```

Alles anzeigen

1.2 Parameter für Materialien

Zum <material_type> kommen wir gleich, vorher ein kurzer Blick ins Innenleben:
- compressionAllowed kann mit den Werten *true* oder *false* belegt werden und bestimmt, ob die Textur intern (im Speicher der Grafikkarte - damit ist nicht diese RLE-Kompressionsgeschichte der TGA's gemeint) komprimiert werden darf oder nicht. Die Kompression kann unter Umständen zu einem Qualitätsverlust führen, spart aber Videospeicher. Wenn man also größere Texturen verwendet, ist dies denk ich mal definitiv eine Option.

- fileName gibt den Pfad zur Textur an sich an und wird in Gänsefüßchen (") gesetzt. Die Angabe erfolgt in Relation zum Ordner ...Train Fever/res/textures.

- **magFilter** und **minFilter** kümmern sich ums Mipmapping. Einerseits behebt Mipmapping den *Scintilliation*-Effekt (eine Art Aliasing beim rendern von Texturen - insbesondere dann, wenn das Objekt sehr klein dargestellt wird (eine 2x2k Textur und das Objekt wird nur 100x100Pixel dargestellt, die "Texturschrumpfung" sieht gerade in Bewegung oft unschön aus)) und durch die Art des Behebens auch noch ein paar Performance-Probleme. Wenn Objekte nur sehr klein dargestellt werden, dann braucht man eigentlich auch keine große Textur - Mipmapping erstellt intern kleinere Versionen der Textur und lädt nur diese je nach Größe. Stellt es euch vor wie ein Texturen-LoD-System. Der Vorteil ist, dass Performance eingespart wird, weil für 100x100Pixel nicht trotzdem 2x2k Pixel gelesen werden müssen, sondern nur 128x128 bspw, Nachteil ist allerdings ein um ca 1/3 höherer Speicherverbrauch, den die kleineren Texturversionen beanspruchen.
Als Wert habe ich hier <mode> angegeben. In OpenGl gibt es fürs Mipmapping folgende Modi:

- GL_NEAREST (wählt schlicht den Farbwert des nahegelegensten Pixels* (nearest neighbor filtering) der Haupttextur (base Mipmap) (entspricht deaktiviertem Mipmapping))
- GL_LINEAR (das gleiche wie NEAREST, nur dass hier die Farbwerte der umliegenden Pixel* gewichtet interpoliert werden)
- GL_NEAREST_MIPMAP_NEAREST (hierbei wird die nächste Mipmap gewählt** und nearest neighbor filtering angewandt)
- GL_NEAREST_MIPMAP_LINEAR (hierbei wird zwischen den nächsten Mipmaps interpoliert** und nearest neighbor filtering angewandt)
- GL_LINEAR_MIPMAP_NEAREST (hierbei wird die nächste Mipmap gewählt** und lineare Interpolation angewandt)
- GL_LINEAR_MIPMAP_LINEAR (hierbei wird zwischen den nächsten Mipmaps interpoliert** und lineare Interpolation angewandt)

Spoiler anzeigen

Als Anmerkung sei noch gesagt, dass ihr diese Werte OHNE den GL_ Präfix hier angeben müsst. Also nicht GL_LINEAR_MIPMAP_LINEAR sondern LINEAR_MIPMAP_LINEAR! Zudem gehört auch diese Angabe wieder in Gänsefüßchen ("). **magFilter** (magnification - Vergrößerung) gibt hierbei die zu nutzende Methode an, wenn die eigentliche Textur vegrößert werden muss (man ist nah rangezoomed und das Objekt wird bspw. mit 500Pixeln dargestellt, obwohl die Textur dafür nur 400Pixel groß ist) und **minFilter** (minification - Verkleinerung) eben, wenn sie verkleinert werden soll. Für den **magFilter** kann man üblichweise auf Mipmapping verzichten, da in diesem Falle sowieso das base-level (die größte = originale Textur) angezeigt wird und es kein noch größeres MipLevel gibt.

- **wrapS** und **wrapT** sind ähnlich zu handhaben. Auch hier greift <wrapping> auf OpenGL-Modi zurück:

- GL_REPEAT (Textur wird wiederholt)
- GL_CLAMP (Textur wird nicht wiederholt)
- GL_CLAMP_TO_EDGE (Textur wird nicht wiederholt und ggf. mit der "letzten" gültigen Farbe aufgefüllt)
- GL_CLAMP_TO_BORDER (Textur wird nicht wiederholt und ggf. mit einer definierten "Border-Color" aufgefüllt)

Ich gebe zu, so richtig werd ich jetzt auch nicht daraus schlau (mein OpenGL-Buch ist englisch und das klingt irgendwie alles gleich xD), empfohlen wird jedenfalls CLAMP_TO_EDGE. wrapS und T steht dabei für die Texturkoordinaten S und T, einer Texturkoordinaten-Ensprechung für XY - S wäre also nichts weiter wie X in "Texturensprache" und T eben Y. Auch diese Angabe erfolgt wieder ohne GL_ Präfix und muss in Gänsefüßchen gefasst werden (").

- **type** gibt den Texturtyp an. Gängig wäre 2D, der Wert <type> hierfür wäre "TWOD" - two dimensional. OpenGL unterstützt prinzipiell auch 3D-Texturen (Nebel beispielsweise), ob das für uns aber auch nutzbar ist, kann ich nicht sagen. Ich tippe jedenfalls darauf, dass es dann "THREED" lauten müsste. Eine weitere Möglichkeit wären Cube-Maps für Skyboxen. Hier müsste man dann dementsprechend "CUBE" angeben.

- **mipmapAlphaScale** sucht derweil nach einem Erklärbär - auf gut deutsch: Hier hab ich nun wirklich keine Ahnung. Ihr seht schon, das Beste hab ich mir zum Schluss aufgehoben.

1.3 Mögliche Materialien-Typen

Nun aber zurück zu den <material_type>'s. Hier gibt es verschiedene Möglichkeiten und Kombinationen für euch. Diese werden nicht in Gänsefüßchen (") gesetzt, Ich will sie im Folgenden alle erst einmal nennen und ihre Auswirkungen zusammen fassen:

- **map_color_reflect**

- RGB-Kanäle: Farbe/Textur
- Alpha-Kanal: Grad der Spiegelung

- **map_color_alpha**

- RGB-Kanäle: Farbe/Textur
- Alpha-Kanal: Transparenz

- **map_normal**

- RGB-Kanäle: Vektorielle Nutzung, Einheitsvektoren zur Bestimmung der Abweichung von der Flächennormalen
- Alpha-Kanal: Specular-Gradient

- **map_env**

- Umgebungs-Textur, Alpha-Kanal erscheint mir hier überflüssig

1.4 Anmerkungen

map_color_reflect ermöglicht es euch spiegelnde Flächen darzustellen. Hierbei gilt zu beachten: Je transparenter, desto spiegelnder. Als Anwendungsgebiet wären hier Fenster zu nennen. Wer nicht gerade (wie ich ^^) auch das Innenleben eines Fahrzeuges/Hauses modellieren möchte, kann mit dieser Methode gute Ergebnisse erzielen.
map_color_alpha ermöglicht es euch transparente Flächen darzustellen. Der Transparenzgrad gibt natürlich die "Durchsichtigkeit" (eben Transparenz) an. Sollte soweit logisch sein. Anwendungsgebiete wären hier meinetwegen irgendwelche Stahlmasten. Anstelle eines komplexen Meshes knüppelt man eine entsprechende Stahlstreben-Textur einfach über ein simples 4-Eck Mesh. Zäune sind somit auch gut machbar. Oder eben für so verrückte wie mich: tatsächlich auch Glasscheiben 8) Image not found or type unknown
map_env gibt eine Umgebungstextur an (environment). Sie wird daher üblicher-/sinnvollerweise als Cubemap deklariert. Hier könnt ihr also ggf auch eine eigene Umgebungstextur beilegen, ansonsten nutzt ihr einfach die von TF: "c.tga".
map_normal ermöglicht euch die Nutzung von Normalmaps. Der Alpha-Kanal ist für specularity zuständig - also den Glanzfaktor meinetwegen. Holz zum Beispiel sieht eher matt aus, hier wäre also keine Transparenz sinnvoll. Chrom/Stahl... hingegen glänzt ganz gut in der Sonne - Transparenz wäre also durchaus eine Option. Falls ihr fragen zu Normalmaps an sich habt, schreibt es - fürs erste lasse ich eine Detailerklärung aussen vor. Bei Interesse ergänze ich es aber gern.

Weiterhin sollte klar sein, dass eine Umgebungstextur nur Sinn macht, wenn man spiegelnde Flächen/Materialien nutzt. Normalmaps mit specularity sollten aber sowohl bei ..._reflect wie auch bei ..._alpha wirken.

2 Der type-Parameter

Fast am Ende unserer .mtl-Datei befindet sich noch ein type-Parameter. Diesen müsst ihr je nach verwendeten Eigenschaften anpassen:

- type = "REFLECTIVE", (ihr verwendet map_color_reflect, ob hierbei zwingend ein map_env Eintrag benötigt wird entzieht sich meiner Kenntnis. es ist durchaus möglich, dass ein Fehlen mit der default-Textur gefüllt wird)
- type = "REFLECTIVE_NRML_MAP", (ihr verwendet map_color_reflect und zusätzlich map_normal, map_env wie eben)
- type = "TRANSPARENT", (ihr verwendet map_color_alpha)
- type = "TRANSPARENT_NRML_MAP", (ihr verwendet map_color_alpha und zusätzlich map_normal - selbst bisher nicht genutzt, ist eher geraten das es das gibt)