Materials

Inhaltsverzeichnis

- 1 Structure of a .mtl-file
  - 1.1 Structure of material-type declarations
  - 1.2 Parameters for materials
  - 1.3 end of translation for now
  - 1.4 Mögliche Materialien-Typen
  - 1.5 Anmerkungen
- 2 Der type-Parameter

In TrainFever there are different file formats (german) wich answer their special purposes. Here i want do take a closer look at the .mtl material files.

1 Structure of a .mtl-file

An .mtl-files basic set-up looks like this:
Code

```
function                                                                    data()
return                                                                           {




        },

        [...]




            },

        },


        },

    },

}
end
```

Alles anzeigen

1.1 Structure of material-type declarations

Here i cut something in the mid [...] because this part shall be the core of this entry: The different possibilities of material-types and their setting. As far as I know, this can be a mix from up to three blocks, each having a structure like that:
Code

```
<material_type>                                =                                {









        },
```

Alles anzeigen

1.2 Parameters for materials

I'll hav a closer look at the <material_type> later, for now i want to show the inside of this block:
- **compressionAllowed** accepts the values *true* or *false* and determines, if a texture gets compressed internally (meant is the data in the video ram of your GPU, not a file compression on your disk like RLE) or not. This compression could probably lead to a loss in quality but saves video ram. While using big textures this could be a good option.

- **fileName** wants to be fed with the path to your texture in relation to this folder: ...Train Fever/res/textures. You need to surround it with quotes (").

- **magFilter** and **minFilter** take care of Mipmapping. On the one hand mipmapping helps against the scintilliation-effect
(some kind of aliasing while rendering textures - especially when an object is drawn very small while having a big texture attached (lets say a 2x2k texture has to be shrinked to 100x100 pixels, then it could look vary ugly, especially when movin around) and on the other hand some performance issues. When objects are very small on the screen you dont need a big texture - but when you see it zoomed in, you'll do. To solve this, mipmapping generates a bunch of texture copies, each a bit smaller then the last one. Then it is able to provide small objects with a small version of your texture and zoomed objects are drawn with the big original texture. Imagine it like a "texture-LoD". The advantage is, that you save some performance, because to display 100x100 pixels, you dont have to read the whole 2x2k texture from your memory (but lets say a version with 128x128 pixels). The disadvantage is a higher memory-consumption. In default OpenGL shrinks each miplevel by half the size (2x2k -> 1x1k...) and with mipmaps you end up in about one third more memory used.
I gave <mode> as value here and in OpenGl there are the following modi for mipmapping:

- GL_NEAREST (simply chooses the color of the nearest pixel* (nearest neighbor filtering) from the main texture (base Mipmap) (at least its deactivated mipmapping))
- GL_LINEAR (the same like nearest NEAREST, only the colorvalue is interpolated between the surrounding pixels*)
- GL_NEAREST_MIPMAP_NEAREST (the nearest mipmap (miplevel) is choosen** and nearest neighbor filtering is applied)
- GL_NEAREST_MIPMAP_LINEAR (the values of the nearest miplevels will be interpolated** and nearest neighbor filtering is applied)
- GL_LINEAR_MIPMAP_NEAREST (the nearest mipmap (miplevel) is choosen** and linear interpolation is applied)
- GL_LINEAR_MIPMAP_LINEAR (the values of the nearest miplevels will be interpolated** and linear interpolation is applied)

Spoiler anzeigen

1.3 end of translation for now

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.

1.4 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_reflectRGB-Kanäle: Farbe/Textur

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

Alpha-Kanal: Transparenz
- map_normalRGB-Kanäle: Vektorielle Nutzung, Einheitsvektoren zur Bestimmung der Abweichung von der Flächennormalen

Alpha-Kanal: Specular-Gradient
- map_envUmgebungs-Textur, Alpha-Kanal erscheint mir hier überflüssig

1.5 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
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)