# Rückkonvertierung TPF2 -> TPF1 II: Die .mdl

Diese Erklärung setzt da ein, wo der vorherige Artikel über die Konvertierung der einzelnen Dateien aufhört.

Nach den in diesem Artikel beschriebenen Schritten sind die diversen Unterdateien, Meshes und Materialien des Modells für TPF1 verständlich gemacht. Die große .mdl, in der aus den Einzelteilen das Modell wird, fehlt aber noch. Also nehmen wir uns jetzt diese auch noch vor.

Ich werde hier beim MAN-Haubendiesel bleiben; das hier gezeigte Beispiel wird die .mdl des Betonmischers sein, die sowohl in TPF2 als auch in 1 vorhanden ist. Es ist also möglich, die beiden direkt zu vergleichen und so die Änderungen vorzuführen.

Grundsätzlich ist die .mdl in TPF1 wesentlich einfacher aufgebaut als in TPF2. Das liegt daran, daß viele Informationen, die in 2 in der .mdl stehen, in 1 in den einzelnen Unterdateien eingetragen werden müssen - deswegen wurden im vorherigen Artikel zahlreiche Erweiterungen der Spieldateien vorgenommen. Hier wird es andersherum funktionieren, hier werden einige Punkte aus der 2-.mdl herausfliegen, nur wenige hinzukommen und einzelne auch umbenannt werden müssen.

Der Kopf der .mdl ist grundsätzlich einmal identisch. Wo es in 2 heißt:

Code

```
1. local vec3 = require "vec3"
2. local transf = require "transf"
3. function data()
4. return {
5. boundingInfo = {
6. bbMax = { 3.85026, 1.37978, 2.78063, },
7. bbMin = { -4.0266, -1.37978, -0.00589, },
8. },
9. collider = {
10. params = { },
11. transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, },
12. type = "MESH",
13. },
```

Display More

...steht in 1:

Code

```
1. local vec3 = require "vec3"
2. local transf = require "transf"
3. function data()
```

```
4. return {
5. boundingInfo = {
6. bbMax = { 3.85026, 1.37978, 2.78063, },
7. bbMin = { -4.0266, -1.37978, -0.00589, },
8. },
9. collider = {
10. params = {
11. },
12. type = "MESH",
13. },
```

Display More

Der Block "collider = '' wird also ausgeleert, das ist alles.

**Meshes und Gruppen**

Der nächste Schritt wird etwas komplizierter, denn jetzt kommt die Auflistung der einzelnen Meshes und Gruppen.

In TPF2 ist der Anfang so:

Code

```
1. lods = {
2. {
3. node = {
4. animations = { },
5. children = {
```

In 1 fehlt die Ebene "node = {" und der Block "animations= {},". Es sieht also so aus:

Code

```
1. lods = {
2. {
3. children = {
```

Wir merken uns bitte, daß in 2 vier Schweifklammern geöffnet stehen, in 1 derer nur drei. Die müssen wir nachher alle wieder an der richtigen Stelle schließen.

Als nächstes kommt eine ganze Menge von Blöcken, die die einzelnen Teile und Gruppen aufrufen. Teile sind relativ simpel, bei Gruppen muß man aber aufpassen: die haben wir im vorherigen Artikel von Hand neu erstellen müssen und werden sie jetzt hier in der .mdl aufrufen.

In TPF2 sieht so ein Block so aus:

Code

```
1. {
2. _meshId = 1,
3. _origMeshId = 1,
4. materials = { "vehicle/truck/MAN415-Betonmischer.mtl", },
5. mesh = "vehicle/truck/MAN415/MAN415_Fahrerhaus.msh",
6. name = "MSH_MAN415_Fahrerhaus_1",
7. transf = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, },
8. },
```

Da stehen viele Dinge drin, die wir schon in die .msh-Dateien für TPF1 reingeschrieben haben oder die in 1 keine Entsprechung haben. Die werden wir hier also einfach rausnehmen.

Derselbe Block sieht in TPF1 so aus:

Code

```
1. {
2. id = "vehicle/truck/MAN415/MAN415B_Fahrerhaus.msh",
3. transf = {
4. 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
5. },
6. type = "MESH",
7. },
```

Der Aufruf der .msh heißt hier nicht "mesh=", sondern "id = ", das müssen wir ändern. Die Transformationsmatrix ist identisch, kann also 1:1 übernommen werden. Und daß das ein Mesh ist, müssen wir TPF1 auch nochmal sagen, also kommt "type = "MESH", " hinzu. Alles andere fliegt raus.

Wenn wir eine Gruppe aufrufen wollen (wir erinnern uns: in TPF2 erkannten wir die an einem Unterblock, der wieder mit "children = " begann und wir haben das in eine separate .grp auslagern müssen), dann geht das fast genauso:

Code

```
1. {
2. id = "vehicle/truck/MAN415B_Kardanwelle2.grp",
3. transf = {
4. 0.96866, -0.0, 0.24838, 0.0, 0.0, 1.0, 0.0, 0.0, -0.24838, 0.0, 0.96866, 0.0, 0.39598, -0.0, 0.96, 1.0,
5. },
6. type = "GROUP",
7. },
```

Die einzigen Unterschiede sind, daß die in "id = " genannte Datei jetzt ".grp" heißt und daß unter "type = " jetzt "GROUP" einzutragen ist. Die Transformationsmatrix kann wieder 1:1 aus der TPF2-.mdl übernommen werden und braucht keine Änderung. Wenn die Gruppe animiert wird, dann stehen die Keyframes schon in der .grp drin; aufgerufen werden sie erst später in der 1-.mdl.

Und wenn wir alle .msh und .grp so abgehandelt haben, folgt eine weitere schließende Schweifklammer, die den Block "children = {" abschließt. Weiter geht es mit den Animationen.

**Aufrufe der Animationen**

Das funktioniert in TPF1 etwas anders als in 2. Was in 2 direkt bei den Meshes stand, bekommt hier einen eigenen Block: events = {.

Auch wenn es keine Animationen gibt, muß er vorhanden sein. Auf jeden Fall kommt also in die .mdl hinein:

Code

```
1. events = {
2. },
```

...und dazwischen kommen dann die Animationen. Der Auslöser der Animationen steht in den 2-.mdl genau so drin, wie er hier hineinkommt.

Code

```
1. drive = {
2. },
```

In diesen Block kommen dann die Laufnummern der zu animierenden Teile, der Name der Animation selbst und eine Information, ob die Animation vorwärts oder rückwärts abzuspielen ist. Wenn wir das erste Mesh der obigen Liste animieren wollen, die Animation "MAN415_Kardanwelle" heißt (das steht in der .msh drin, da kann man nachsehen) und die Animation vorwärts laufen soll, dann sieht der Block so aus:

Code

```
1. [1] = {
2. forward = true,
3. name = "MAN415_Kardanwelle",
4. },
```

Hier muß man ein wenig aufpassen, das Spiel ist hier gegen Fehler sehr intolerant. Es erwartet, im ersten Mesh der obigen Liste eine Animation mit dem genannten Namen vorzufinden, die auch legale Keyframes

enthalten muß. Jeder Fehler hier wird einen völlig undokumentierten CTD verursachen - ich empfehle also, diese Animationsaufrufe einen nach dem anderen einzubauen und erst, wenn einer funktioniert, den zweiten hinzuzufügen.

Ein fertiger solcher "events = "- Block sieht aus wie folgt:

Code

```
events = {
drive = {
[3] = {
forward = true,
name = "MAN415_KardanwelleAction",
},
[4] = {
forward = true,
name = "drive",
},
[5] = {
forward = true,
name = "MAN415_Kardanwelle_3Action",
},
},
forever = {
[17] = {
forward = true,
name = "forever",
},
},
},
```

Display More

Als nächstes folgt noch ein TPF1-Spezifikum, die Materialkonfiguration. Dieser Block sieht so aus:

Code

```
matConfigs = {
{
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
},
},
```

Für jedes Mesh oder jede Gruppe, die oben aufgelistet ist, muß eine 0 da stehen. Wenn es eine zuviel ist, ist es nicht schlimm, nur eine zuwenig darf es nicht sein.

Die drei Linien "static = false, visibleFrom=0, visibleTo = 100," dürfen hier auch nicht fehlen - die hier

aufgelisteten Teile werden mit den gegebenen Werten von 0-100m Kameraentfernung sichtbar sein.

Es folgt eine schließende Schweifklammer, die den ersten LoD-Block abschließt, und dann gegebenenfalls für den nächsten LoD eine weitere öffnende Schweifklammer, die eine weitere Sektion einleitet, die wiederum genauso aufgebaut ist wie die obige Sektion.

Zum Schluß folgt eine weitere schließende Schweifklammer, dann sind die Meshes, Gruppen und Animationen erledigt.

Alles weitere ist 1:1 zu übernehmen. Nur die Funktionen, die in TPF1 nicht aktiv sind, sollten herausgenommen werden: die variablen Lichteffekte, die verwischten Propeller und Ähnliches sind in TPF1 nicht integriert. Diese Zeilen kann man einfach löschen.

Und jetzt sollte die Konvertierung abgeschlossen sein.