

OMSI 1.00
Technisches Handbuch –
Software Development Kit (SDK)

Teil 2: Szenerieobjekte
- UNVOLLSTÄNDIG -

Stand: 14. Mai 2011

Inhaltsverzeichnis

EINLEITUNG	3
LEKTION 1: DAS ERSTE EIGENE SZENERIEOBJEKT	4
1.1. Übersicht Dateitypen	4
1.2. Blender-Export der x-Datei	5
1.3. Konvertierung in das o3d-Format	9
1.4. Anlegen einer neuen sco-Datei	9
1.5. Friendly-Name und Gruppen	11
1.6. Texturverzeichnis	12
1.7. Materialeigenschaften im x-Format	13
1.8. „Kantige“ und „weiche“ Objekte	16
1.9. Physikalische Eigenschaften	21
1.10. Kollisionseigenschaften	24
LEKTION 2: MATERIALEIGENSCHAFTEN	27
2.1. Einführung / Theorie	27
2.2. [matl], [matl_change] und [matl_allcolor]	30
2.3. Nightmap, Lightmap, Envir-Map, Bumpmap	36
2.4. Verwendung des Alphakanals als Transparenz	40
2.5. Weitere Möglichkeiten der Alphakanalnutzung	42
2.6. Animation der Texturkoordinaten	44
2.7. Texturadressmodi	45
2.8. Kein Schreiben in Z-Buffer	47
2.9. FreeTex	47
2.10. Textturen	49
2.11. Eigene Fonts	50

Einleitung

Willkommen zum zweiten technischen Handbuch! Nachdem im ersten Buch erklärt wurde, wie die Splines und Szenerieobjekte mit Hilfe des Karteneditors platziert werden, soll nun erklärt werden, wie eigene Splines und Szenerieobjekte gebaut werden.

Gleichzeitig stellt dies die Grundlage des Busbaus dar, welcher im dritten Handbuch erklärt werden soll: Denn der Aufbau von Szenerieobjekten ist dem der Fahrzeuge recht ähnlich; beide verfügen über Meshs und Texturen und können je nach Bedarf über Sound- und Spezialeffekte, Animationen und auch über Scripts verfügen.

Es sollte also der Grundlagen wegen dieses Handbuch über Szenerieobjekte durchgearbeitet werden, bevor mit dem Bau von Bussen begonnen wird.

Achtung: Dies ist eine unvollständige Vorabversion! Die Arbeit an diesem Handbuch ist nicht abgeschlossen – später soll es deutlich umfangreichere Versionen desselben Handbuches geben!

Lektion 1: Das erste eigene Szenerieobjekt

1.1. Übersicht Dateitypen

Zuerst ein kleiner Exkurs:

Öffnen Sie den OMSI-Ordner, gehen Sie in den „vehicle“- und dort in den Ordner „MAN_F90“.

Hier finden Sie zunächst die Fahrzeugdatei mit der Dateiendung „*.bus“. Da dies etwas irreführend sein kann, ist auch alternativ die Dateiendung „*.ovh“ erlaubt (Omsi-Vehicle).

Außerdem sehen Sie folgende Unterordner: „Model“ (enthält die dreidimensionalen Körper, also die Meshs), „Script“, „Sound“ und „Texture“, welche die entsprechenden Dateien enthalten.

Schauen Sie nun einmal in den „model“-Ordner hinein: Sie finden dort die Datei „model.cfg“ sowie diverse *.o3d-Dateien, welche die eigentlichen Meshs darstellen, also die dreidimensionalen Objekte.

Nun das Ganze bei Szenerieobjekten: Wechseln Sie in das Verzeichnis „\Sceneryobjects\Buildings_MC“. Dort befinden sich zahlreiche *.sco-Dateien, welche jeweils ein Szenerieobjekt beschreiben, im „Model“-Ordner finden Sie jedoch lediglich o3d-Dateien, keine model*.cfg-Dateien!

Schließlich können Sie sich auch noch mal einen der Busse ansehen im Ordner „Vehicles\MAN_SD200“: Hier gibt es eine ganze Reihe von *.bus-Dateien, dazu *.org- und *.hof-Dateien und im Model-Ordner wird es so kompliziert, dass weitere Unterordner angelegt wurden. Auch ist hier gut zusehen, dass die „model.cfg“ nicht unbedingt genau so heißen muss!

In diese auf den ersten Blick recht unübersichtliche Dateienlandschaft wollen wir nun etwas Klarheit bringen:

- **model.cfg:** Diese Datei enthält grundsätzlich alle Informationen, die das 3D-Modell beschreiben: Welche Mesh-Dateien (o3d) werden verwendet, welche Effekte oder Animationen werden hinzugefügt.
- ***.bus/*.ovh/*.sco:** Diese Dateien definieren ein Szenerieobjekt oder Fahrzeug. Sie enthalten allgemeine Informationen – Definitionen, wo sich die Achsen befinden, wie sie gefedert sind, wie das Objekt oder Fahrzeug heißt, welchen Hersteller es hat bzw. in welche Gruppen das Objekt eingeordnet werden soll usw.

Wichtig: Es gibt nun zwei Möglichkeiten: Bei einfachen Objekten kann auf die model.cfg verzichtet werden! In diesem Fall übernimmt die *.sco oder

.bus/.ovh-Datei diese Funktion, d.h. dann werden die Einträge aus der model.cfg in die *.sco/*.bus/*.ovh-Datei integriert!

Oder aber, wenn beides getrennt sein soll, verweist die *.sco/*.bus/*.ovh-Datei mit einem „[model]“-Befehl auf die model.cfg. Durch diesen expliziten Aufruf der model.cfg ist es möglich, dass deren Name auch anders lautet. Für eine bessere Lesbarkeit wird aber dennoch dringend empfohlen, bei „model.cfg“ zu bleiben.

Außerdem gibt es noch folgende weitere Dateitypen:

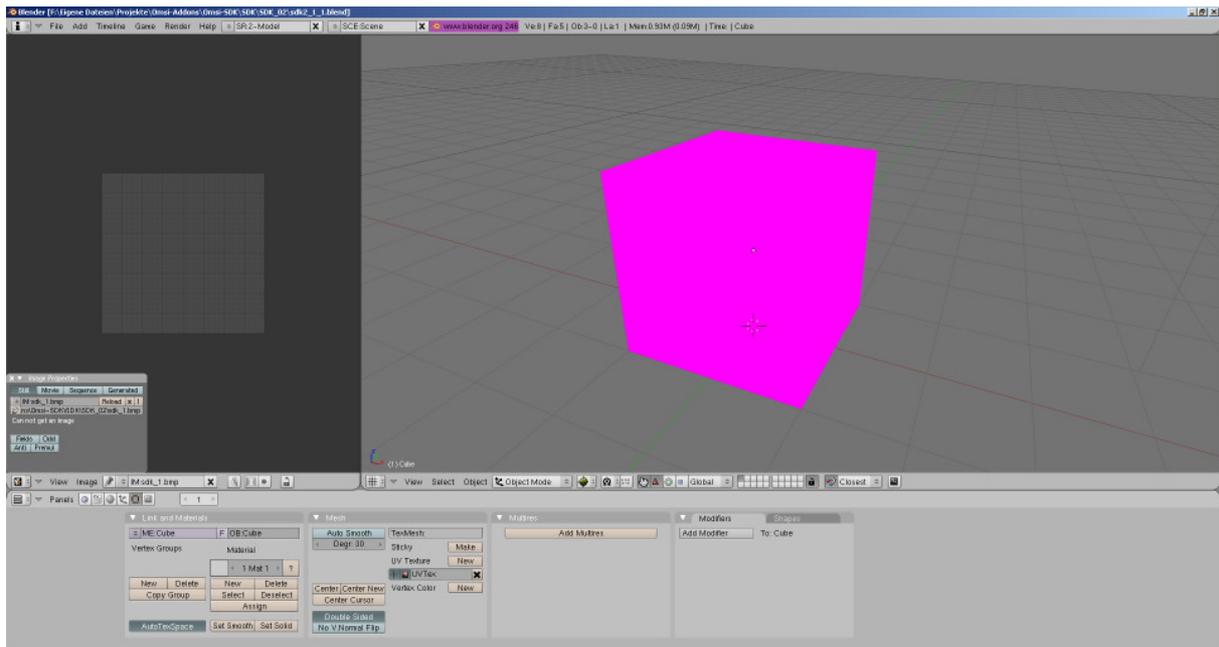
- ***.hof-Dateien:** Wurden bereits im 1. Handbuch vorgestellt
- **Registrierungsdateien:** Meistens *.org, liegen auch im Fahrzeughauptverzeichnis, enthalten Nummernlisten der Fahrzeuge
- **o3d/x-Dateien:** Beschreiben die dreidimensionale Form, das sog. „Mesh“ der Objekte
- **Texturen (*.bmp, *.tga, *.jpg, *.dds, ...)**
- **Scriptdateien (*.osc, *_varlist.txt und *_constfile.txt)**
- **Sounddateien (sound*.cfg, *.wav-Dateien)**
- **Fahrgastraumbeschreibungen (paths*.cfg und passengercabin*.cfg)**

Nach diesem längeren Theorieabschnitt nun die erste Praxis!

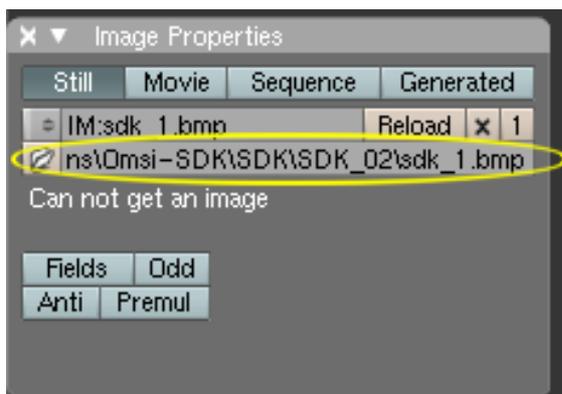
1.2. Blender-Export der x-Datei

Ziel dieses Handbuches soll es nicht sein, das Programm Blender zu erklären; hierzu gibt es bereits unsere Video-Tutorials, außerdem jede Menge Informationen im Internet. Als Hilfestellung finden Sie im SDK die Blender-Objekte, die wir hier verwenden. Sie können aber natürlich auch eigene Blender-Dateien erstellen und mit denen die einzelnen Schritte ausprobieren.

Öffnen Sie nun die Datei „sdk2_1_1.blend“:

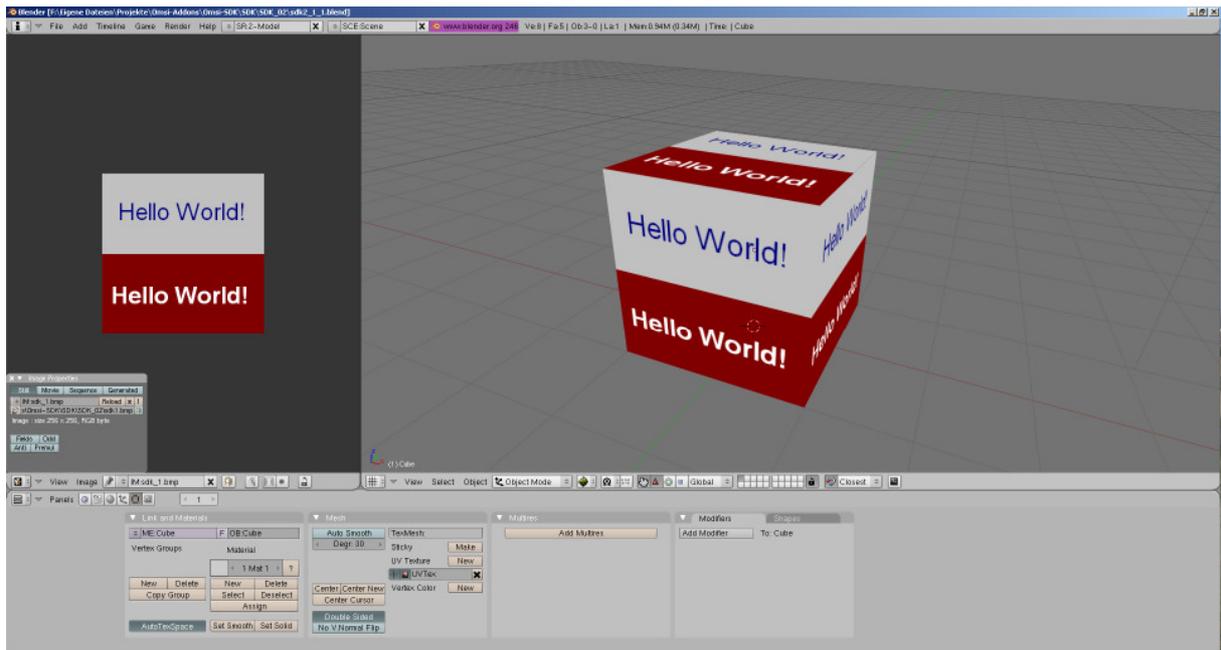


Höchstwahrscheinlich wird der Texturpfad nicht stimmen, weshalb das Objekt in Magenta dargestellt wird. In diesem Fall ändern Sie bitte den Texturpfad an der markierten Stelle so, dass er gültig ist:



Dieses Fenster können Sie ggf. aufrufen, indem Sie [N] drücken während der Mauszeiger über dem Image-Editor schwebt (linkes Teilfenster in diesem Beispiel).

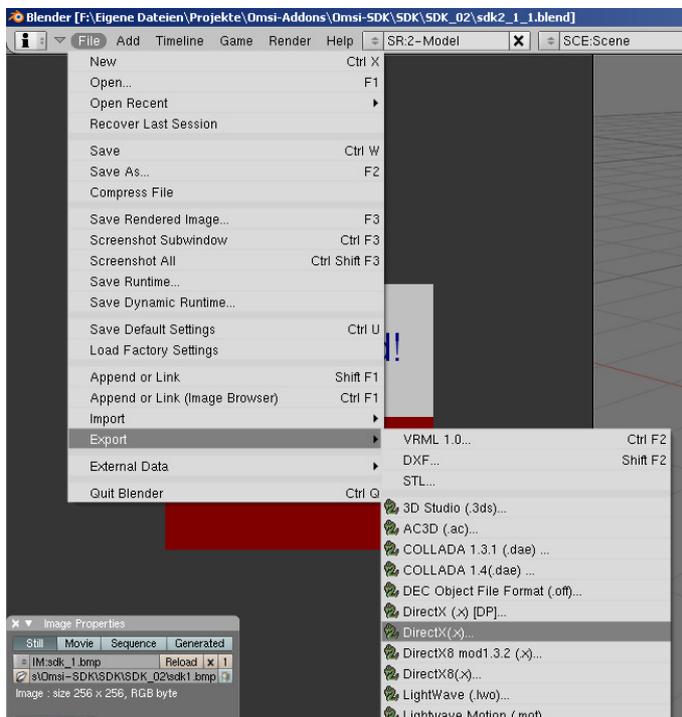
Sobald Sie den korrekten Pfad eingegeben und mit [Enter] bestätigt haben, wird die Textur dargestellt. Gegebenenfalls müssen Sie kurz die Ansicht etwas schwenken, spätestens dann sollte die Textur gezeigt werden:



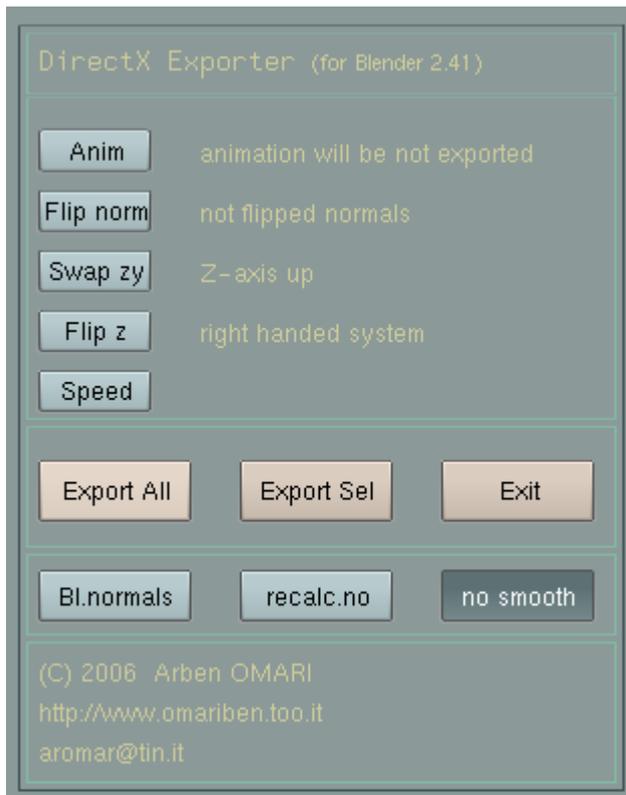
Dies soll also unser erstes Testobjekt sein!

Wir wollen es zunächst als x-Datei exportieren um diese danach mit dem o3d-Konverter in das o3d-Format zu konvertieren.

Klicken Sie im Menü auf „File“ => „Export“ => „DirectX (.x)“:



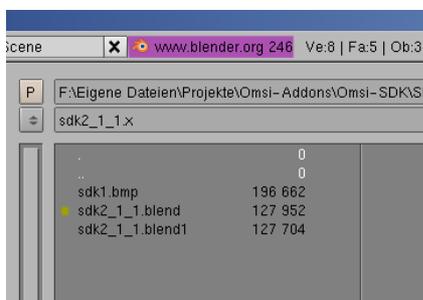
Es erscheint folgendes Fenster:



Damit die x-Datei korrekt für die Konvertierung in das o3d-Format exportiert werden, müssen Sie den Button „Flip z“ abschalten, wie oben gezeigt.

Außerdem müssen Sie stets unterscheiden, ob das Objekt „kantig“ oder „weich“ exportiert werden soll – hierauf wird später eingegangen, erstmal wird nur „kantig“ exportiert – klicken Sie deshalb auf „No Smooth“!

Schließlich können Sie auf „Export All“ klicken, sodass alle vorhandenen Objekte (außer Lichtquellen, Kameras usw.) exportiert werden! Alternativ könnten Sie aber ebenso im Object-Mode die gewünschten Objekte auswählen und „Export Sel“ benutzen! Dann werden nur die ausgewählten Objekte exportiert.



Geben Sie der Datei einen sinnvollen Namen, z.B. „tutorial.x“ und klicken Sie auf „Export DirectX“. Blender sollte nun den Exportvorgang durchführen, sobald Sie den Namen bestätigt haben.

1.3. Konvertierung in das o3d-Format

Die Konvertierung in das o3d-Format (mit Standardoptionen) vollzieht sich sehr einfach: Ziehen Sie einfach die x-Datei auf das Programmsymbol (oder eine Verknüpfung) des Konverters mit dem Namen „OmsiXConv.exe“! Der Konverter wird daraufhin automatisch die Datei mit den Standardoptionen in eine o3d-Datei umwandeln, welche Sie dann mit demselben Namen der x-Datei im selben Ordner finden können!

1.4. Anlegen einer neuen sco-Datei

Nun müssen wir zunächst einmal ein eigenes Objektverzeichnis anlegen – jeder Autor sollte sich ein eigenes Szenerieobjektverzeichnis anlegen, damit nicht die Gefahr besteht, dass sich fremde Objekte durchmischen.

Gehen Sie in den OMSI-Ordner und dort in den Ordner „Sceneryobjects“. Legen Sie nun einen neuen Ordner an mit dem Namen „Tutorial“.

In diesem Ordner legen Sie einen weiteren neuen Ordner an mit dem Namen „model“, der dann also folgenden Pfad haben sollte: „[OMSI]\Sceneryobjects\Tutorial\model“.

Hierin muss die o3d-Datei, die Sie soeben erzeugt haben!

Nun müssen wir noch eine Scenerieobjekt-Datei anlegen. Erstellen Sie hierfür eine neue Textdatei im Ordner „Sceneryobjects\Tutorial“ und geben Sie ihr den Namen „Tutorial_1.sco“.

Wichtig: Hierzu muss in den Ordneroptionen (Windows-Explorer, Menü Extras) auf der Registerkarte „Ansicht“ die Option „Erweiterungen bei bekannten Dateitypen ausblenden“ deaktiviert sein! Sonst können Sie die Dateierweiterung nicht ändern!

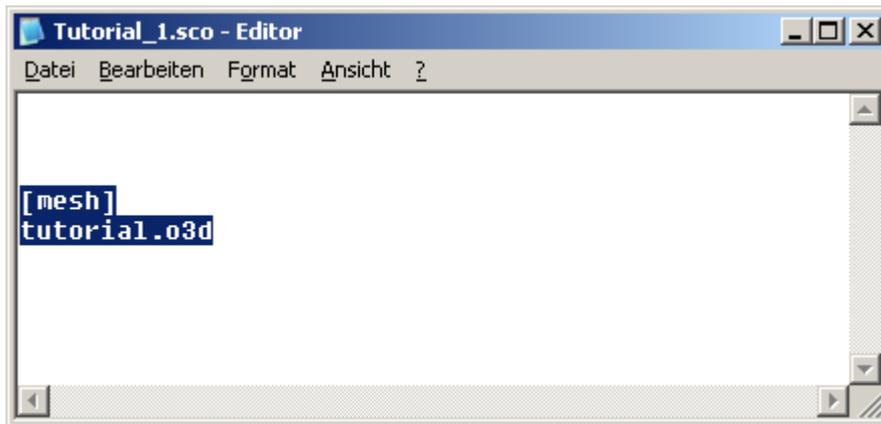
Öffnen Sie die leere Datei im Windows-Editor und schreiben Sie lediglich folgenden Eintrag ein:

```
[mesh]
tutorial.o3d
```

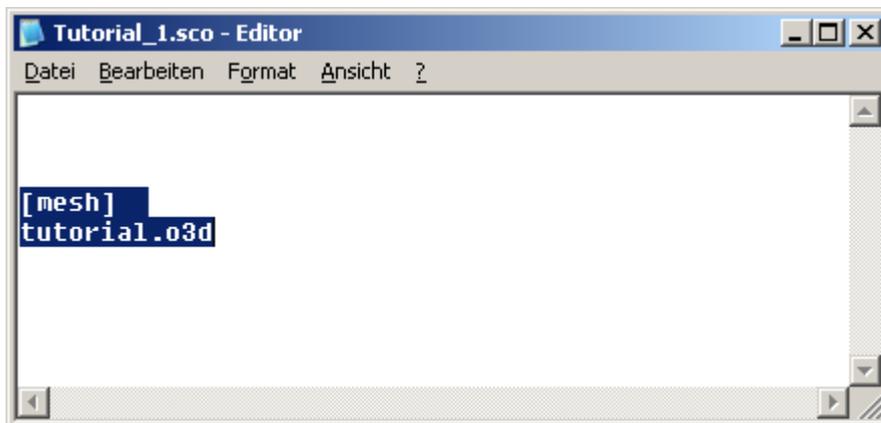
Wichtig: Entscheidend sind bei den OMSI-Konfigurationsdateien folgende Kriterien:

- Groß- und Kleinschreibung beachten! [MESH] ist unzulässig!
- Vor und hinter den relevanten Schlüsselworten (hier: [Mesh]) und Einträgen (tutorial.o3d) dürfen keinerlei andere Zeichen stehen – auch keine Leerzeichen! Dies können Sie ganz einfach testen, indem Sie mit [Strg]+[A] alles markieren:

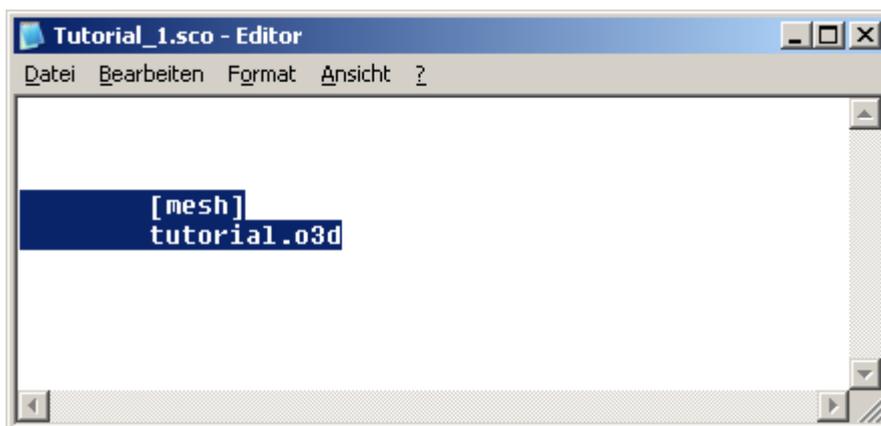
Richtig:



Falsch:

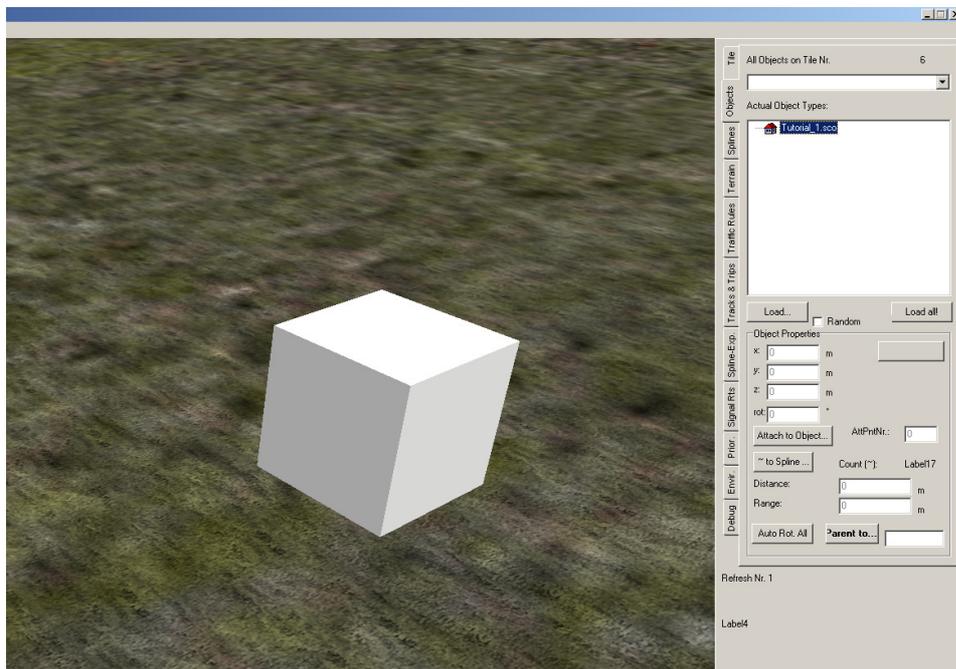


Falsch:



- Alle Bereiche, die nicht durch ein gültiges Schlüsselwort eingeleitet werden, werden ignoriert, sind also Kommentare. Auskommentieren können Sie also, indem Sie genau den oben beschriebenen „Fehler“ begehen: durch Einrücken oder Hinzufügen von Zeichen

Dieses denkbar einfache Szenerieobjekt sollte bereits im Editor erscheinen – allerdings gibt es noch keine Kategorien zum Einsortieren – deshalb erscheint es in der höchsten Hierarchieebene:



Das zweite Problem ist anscheinend, dass die Textur fehlt.

Beide Probleme wollen wir als nächstes lösen!

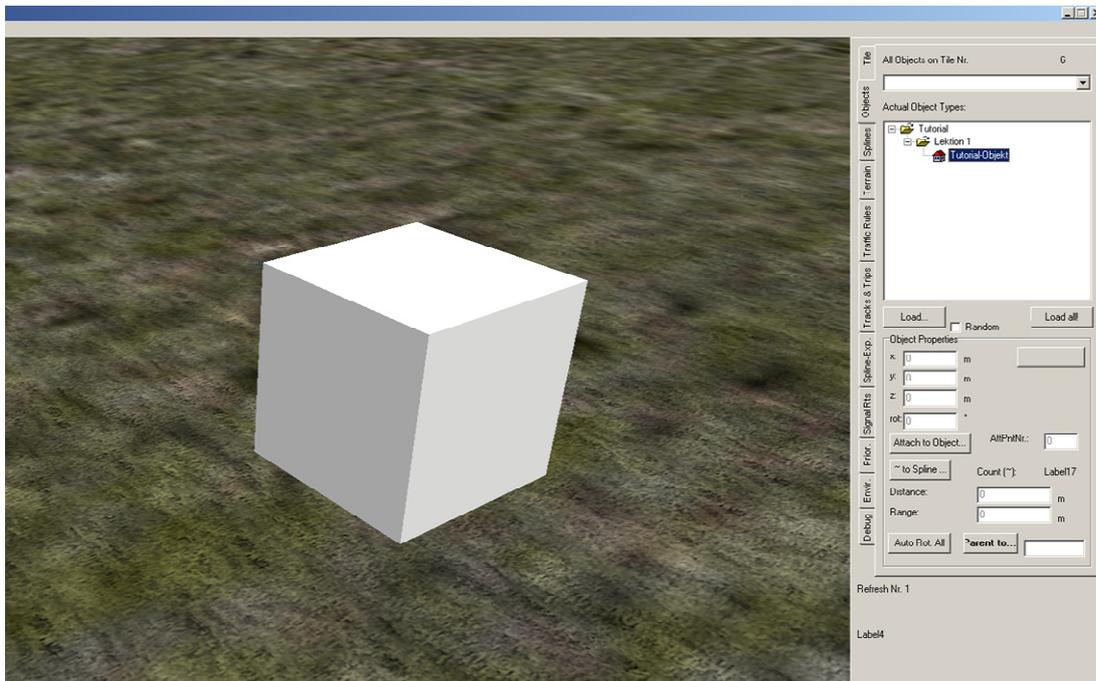
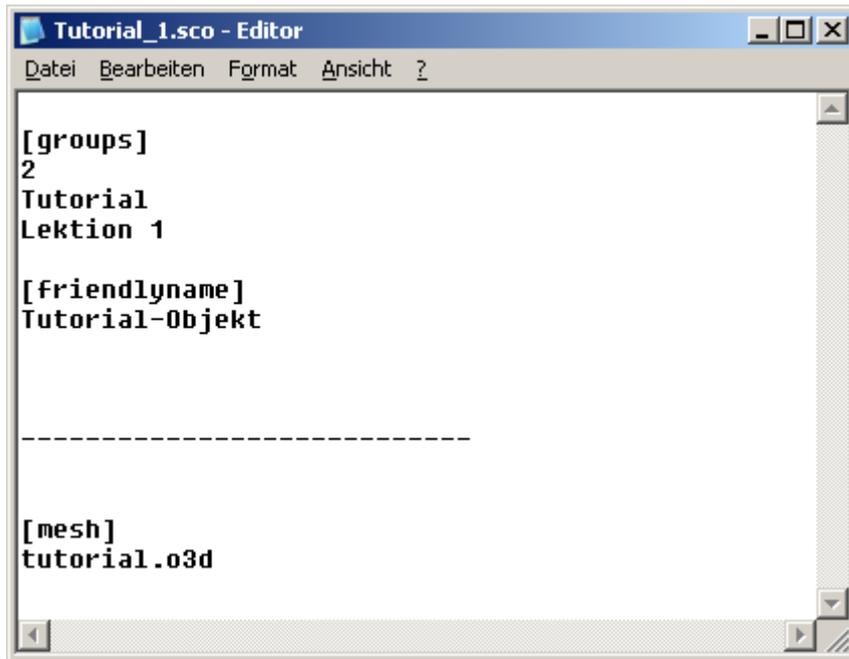
1.5. Friendly-Name und Gruppen

Sie können die *.sco-Datei leicht anpassen, sodass das Objekt – wie auch die anderen Objekte – in der Hierarchie einsortiert wird. Hierzu werden die Kommandos „[groups]“ und „[friendlyname]“ verwendet.

„[friendlyname]“ weist dem Objekt einen „richtigen“ Namen zu, der vom Dateinamen abweichen kann.

„[groups]“ beginnt stets mit der Anzahl der Gruppen bzw. der Hierarchietiefe gefolgt von den Namen der Gruppen. Unser Objekt soll in einer Obergruppe „Tutorial“ und einer Untergruppe „Lektion 1“ einsortiert werden.

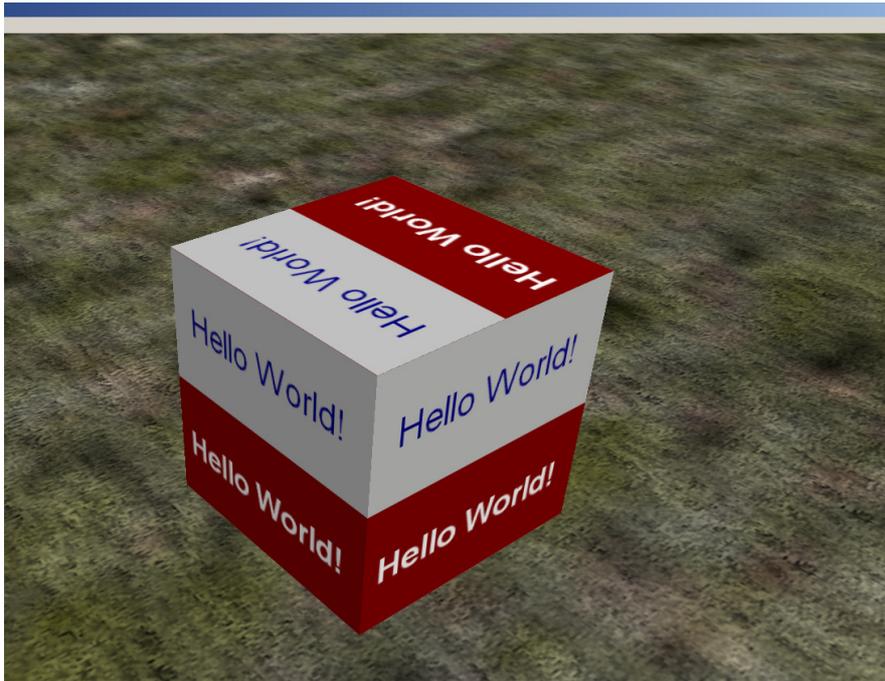
Außerdem habe ich bereits eine Trennlinie eingezogen, um die allgemeinen Objektinformationen (oben) von den speziellen Mesh-Informationen (welche bei komplexen Objekten in der model.cfg eingetragen werden) zu trennen:



Beachten Sie die neue Objekt-Hierarchie!

1.6. *Texturverzeichnis*

Als nächstes soll die Textur ordnungsgemäß eingebaut werden. Legen Sie deshalb im Ordner „Tutorial“ einen Unterordner „texture“ an und kopieren Sie die verwendete Textur „sdk1.bmp“ in diesen Ordner. Starten Sie nun den Editor:



Nun sollte das Objekt also bereits wie gewünscht angezeigt werden!

1.7. Materialeigenschaften im x-Format

Die grundlegenden Materialeigenschaften (*Diffuse*-, *Specular*- und *Emissive*-Farbe sowie der *Power*-Wert) werden direkt in der o3d-Datei gespeichert. Bislang haben wir noch nicht rausgefunden, ob sich diese Werte so in Blender justieren lassen, dass der x-Exporter von Blender diese verwendet. Deshalb manipulieren wir einmal an diesen Werten direkt an der x-Datei!

Öffnen Sie die `template.x` im Windows-Editor. Suchen Sie nach folgender Passage:

```
Material Mat1 {  
    1.0; 1.0; 1.0; 1.0;;  
    1.0;  
    1.0; 1.0; 1.0;;  
    0.0; 0.0; 0.0;;  
    TextureFilename { "sdk1.bmp"; }
```

Beachten Sie die Anordnung: Von Semikolons getrennt finden Sie hier in der ersten Zeile die vier *Diffuse*-Werte (Rot, Grün, Blau und Alpha), welche die Grundhelligkeit des Objektes steuern. Dann folgt ein weiteres Semikolon. In der zweiten Zeile steht der *Power*-Wert, welcher für die Fokussierung des Glanzpunktes zuständig ist. In der dritten Zeile stehen die drei Farbwerte für den Glanzpunkt (*Specular*), in der vierten Zeile die drei Farbwerte, mit denen Sie das Objekt selbstleuchtend einstellen können.

Öffnen Sie die Datei „sdk2_1_2.blend“; diese enthält eine texturierte Kugel. Exportieren Sie diese in die Datei „tutorial.x“, sodass die alte „tutorial.x“ überschrieben wird. Diesmal soll aber „weich“ exportiert werden (und nicht kantig), verwenden Sie deshalb statt „no smooth“ diesmal bitte „Bl.normals“ beim Export!

Konvertieren Sie wie die Datei wie in den vorherigen Kapiteln in das o3d-Format und überschreiben Sie die alte „tutorial.o3d“. Schauen Sie sich die Kugel in OMSI an:



Die Kugel wird diesmal weich (und nicht kantig) gerendert und völlig matt.

Wir wollen nun mit den Materialeigenschaften experimentieren!

Suchen Sie in der neuen „tutorial.x“ wieder den Material-Eintrag – diesmal ist die Datei wesentlich größer, weshalb die Suchfunktion verwendet werden sollte. Die Materialeigenschaften sehen wieder folgendermaßen aus:

```
Material Mat1 {  
    1.0; 1.0; 1.0; 1.0;;  
    1.0;  
    1.0; 1.0; 1.0;;  
    0.0; 0.0; 0.0;;  
    TextureFilename { "sdk1.bmp"; }
```

Offensichtlich stimmt das Bild in OMSI nicht mit diesen Eigenschaften überein! Woran liegt das?

Nun, das Problem ist, dass der Blender-Exporter standardmäßig immer diese sehr untypischen Werte verwendet – insbesondere Power = 1 und Reflection = 1, 1, 1 führt zu sehr unschönen Ergebnissen.

Aus diesem Grund führt der o3d-Konverter eine automatische Korrektur dieser Werte durch, sodass er immer dann, wenn er die oben gezeigten Werte vorfindet, diese durch die folgenden Standardwerte für matte Objekte ersetzt:

```
Material Mat1 {
    1.0; 1.0; 1.0; 1.0;;
    0.0;
    0.0; 0.0; 0.0;;
    0.0; 0.0; 0.0;;
    TextureFilename { "sdk1.bmp"; }
```

Nur, wenn die Werte mit der Hand verändert werden, geht der o3d-Konverter davon aus, dass die neuen Werte auch tatsächlich erwünscht sind.

Ändern Sie nun die Materialeigenschaften auf folgende Werte:

```
Material Mat1 {
    1.0; 0.0; 0.0; 1.0;;
    0.0;
    0.0; 0.0; 0.0;;
    0.0; 0.0; 0.0;;
    TextureFilename { "sdk1.bmp"; }
```

Wichtig: Konvertieren Sie die geänderte „tutorial.x“-Datei erneut und kopieren Sie sie wieder in das entsprechende Verzeichnis im OMSI! Andernfalls werden Sie keine Änderungen feststellen!

Das Ergebnis sieht nun so aus:



Das diffuse Licht der Sonne wird nun mit der Farbe rot multipliziert! Diese Anwendung ist nun allerdings nicht so wirklich sinnvoll – interessanter wird es hingegen, wenn wir die Werte für *Power* und *Specular* anpassen! Probieren Sie nacheinander folgende Werte aus:

```
Material Mat1 {
    1.0; 1.0; 1.0; 1.0;;
    10.0;
    0.9; 0.9; 0.9;;
    0.0; 0.0; 0.0;;
```

```
Material Mat1 {
    1.0; 1.0; 1.0; 1.0;;
    50.0;
    0.9; 0.9; 0.9;;
    0.0; 0.0; 0.0;;
```

```
Material Mat1 {
  1.0; 1.0; 1.0; 1.0;;
  10.0;
  0.5; 0.5; 0.5;;
  0.0; 0.0; 0.0;;
```



Schließlich wollen wir noch den Emissive-Wert ausprobieren: Geben Sie für diesen ein Knallblau ein und verdunkeln Sie die Szene im Editor:

```
Material Mat1 {
  1.0; 1.0; 1.0; 1.0;;
  0.0;
  0.0; 0.0; 0.0;;
  0.0; 0.0; 1.0;;
```

Wichtig: Beachten Sie, dass diese Änderungen an der x-Datei überschrieben werden, sobald Sie die x-Datei aus Blender heraus neu exportieren! Bauen Sie also erst das Objekt ganz fertig und fügen Sie erst danach spezielle Materialeigenschaften hinzu!



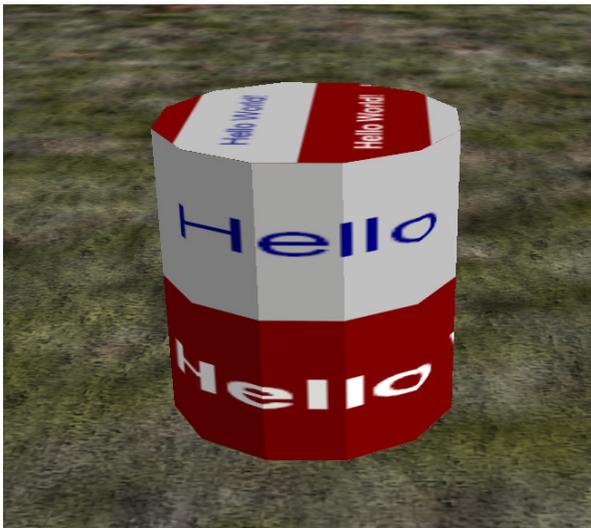
1.8. „Kantige“ und „weiche“ Objekte

Wir wollen nun etwas genauer auf die Möglichkeiten eingehen, Objekte „kantig“ oder „weich“ darzustellen. Probieren Sie mit dem Objekt „sdk2_1_2.blend“ erst noch einmal die Variante „No Smooth“ aus:

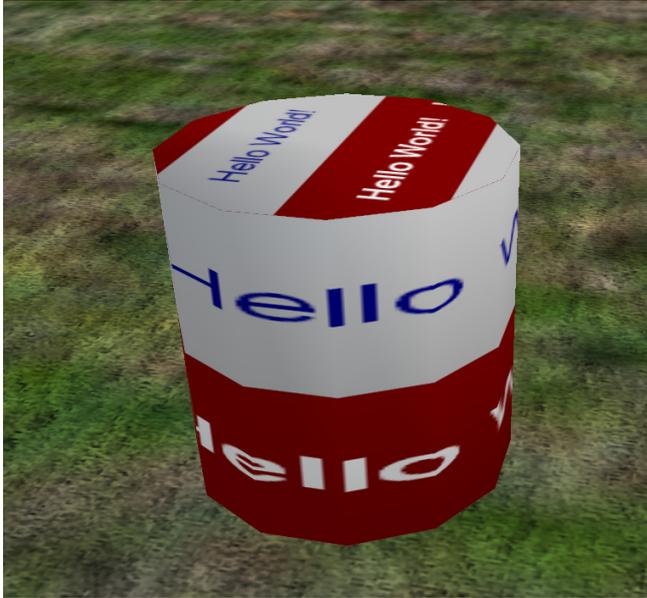


Ganz offensichtlich wird hier jede Fläche separat schattiert, während in der vorherigen Variante die Flächen zusammenhängend weich schattiert werden!

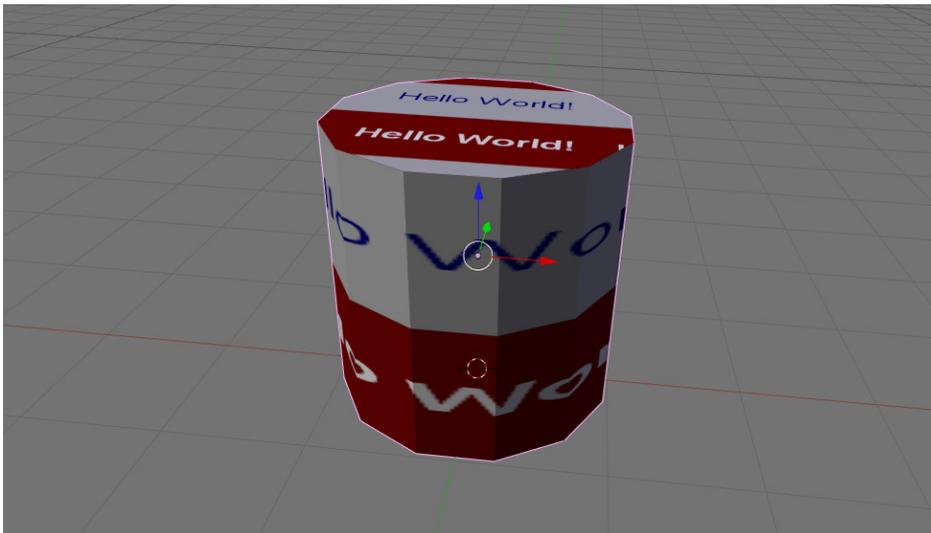
Soweit, so gut. Aber was passiert, wenn *beide* Darstellungsformen gemischt vorkommen sollen? Dies passiert z.B. dann, wenn Sie einen Zylinder bauen. Exportieren Sie deshalb zunächst die Datei „sdk2_1_3.blend“ mit der Option „No Smooth“ und überschreiben Sie damit die alte „tutorial.o3d“:



Zur Not reicht das. Aber wirklich schön sieht es nicht aus... Also exportieren wir einfach mal mit „Bl.Normals“:



Jetzt ist zwar die Zylinderwandung weich – aber auch die „Deckel“ werden weichgezeichnet und „fließen“ in die Seitenwand über – diese Darstellung sieht sehr unrealistisch aus! Bemerkenswerter Weise hat Blender den Zylinder übrigens „weich“ exportiert, obwohl er im „Solid-Mode“ in Blender kantig dargestellt wird:

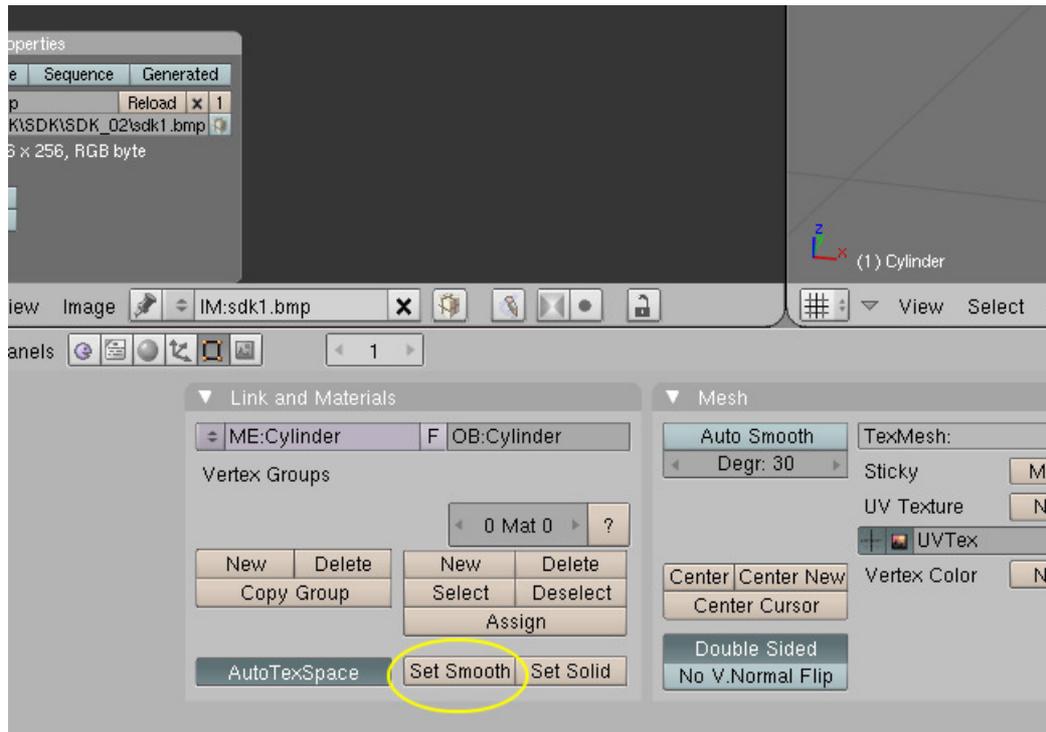


Folgendes ist nun in so einem Fall zu tun:

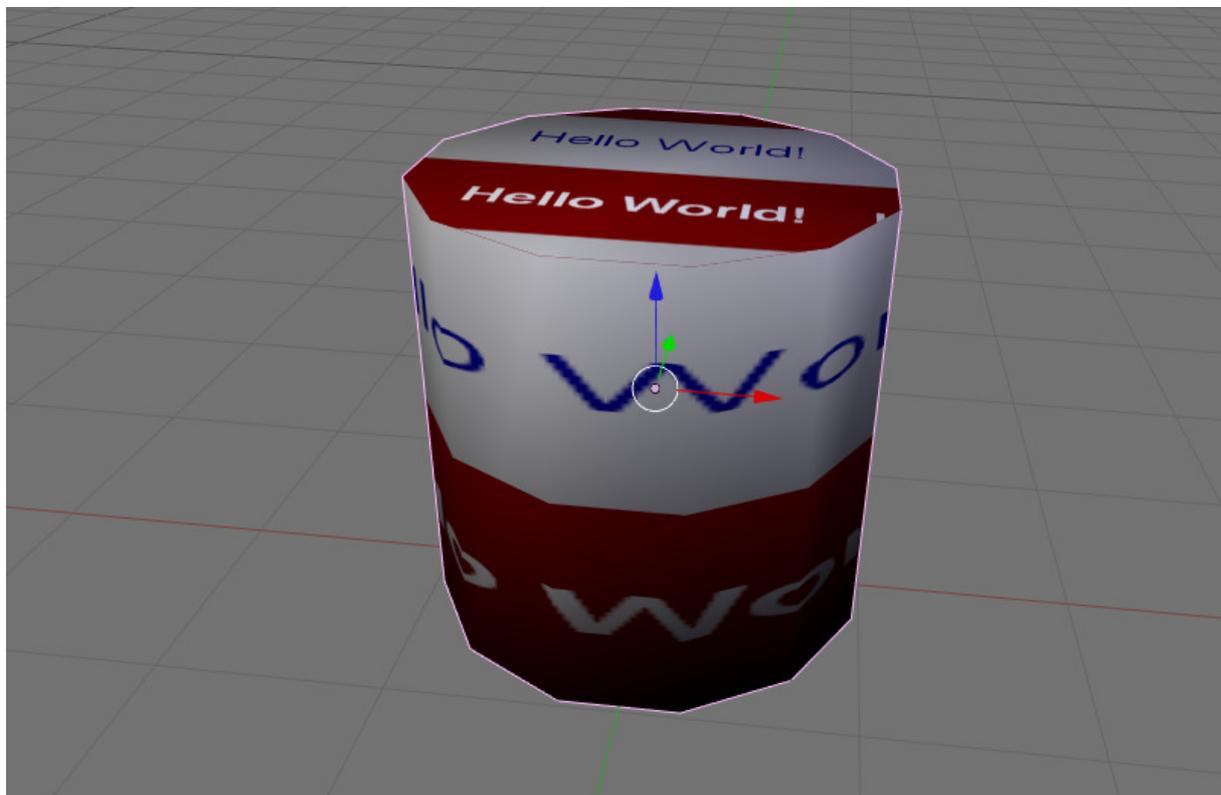
- Umstellen der Blender-Darstellung auf „Weichzeichnung“
- Trennen der Flächen voneinander, sodass die „gewollten“ Kanten auch kantig dargestellt werden
- Exportieren im Modus „Bl.Normals“.

Also Schritt für Schritt:

Gehen Sie in den Objekt-Modus, markieren Sie das (einzige) Objekt und klicken Sie unten unter „Link und Materials“ auf „Set Smooth“:

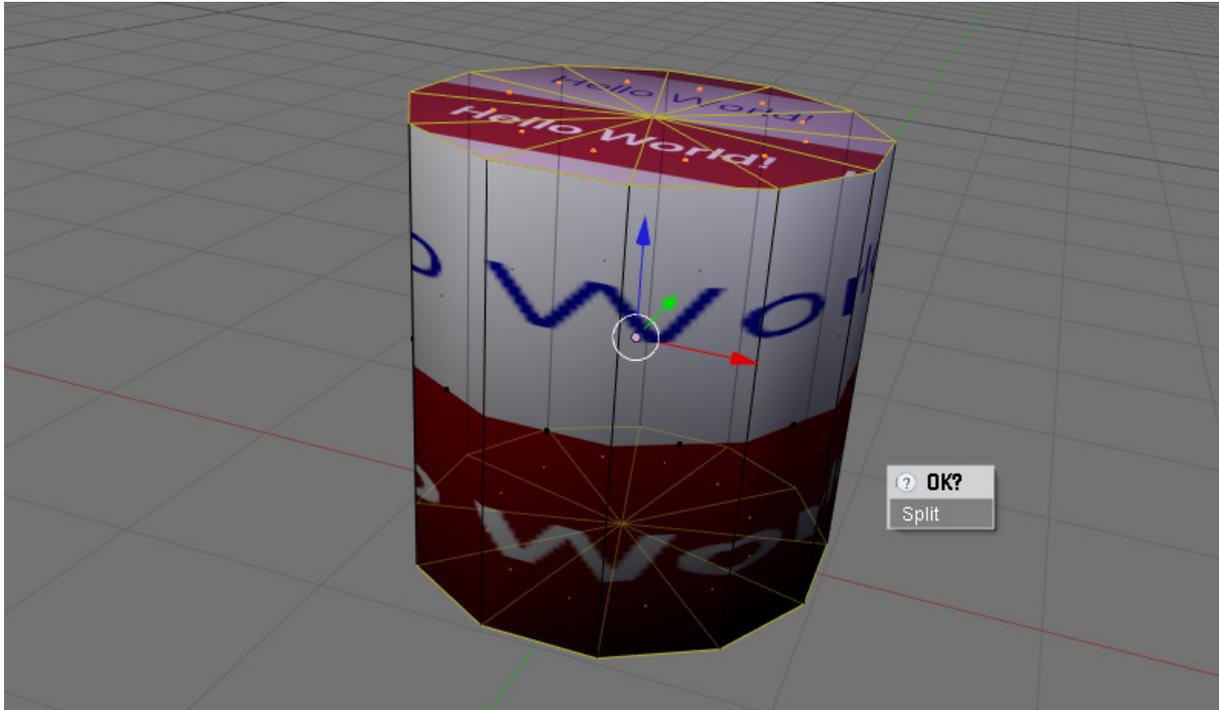


Dann sollte der Zylinder in etwa so aussehen:

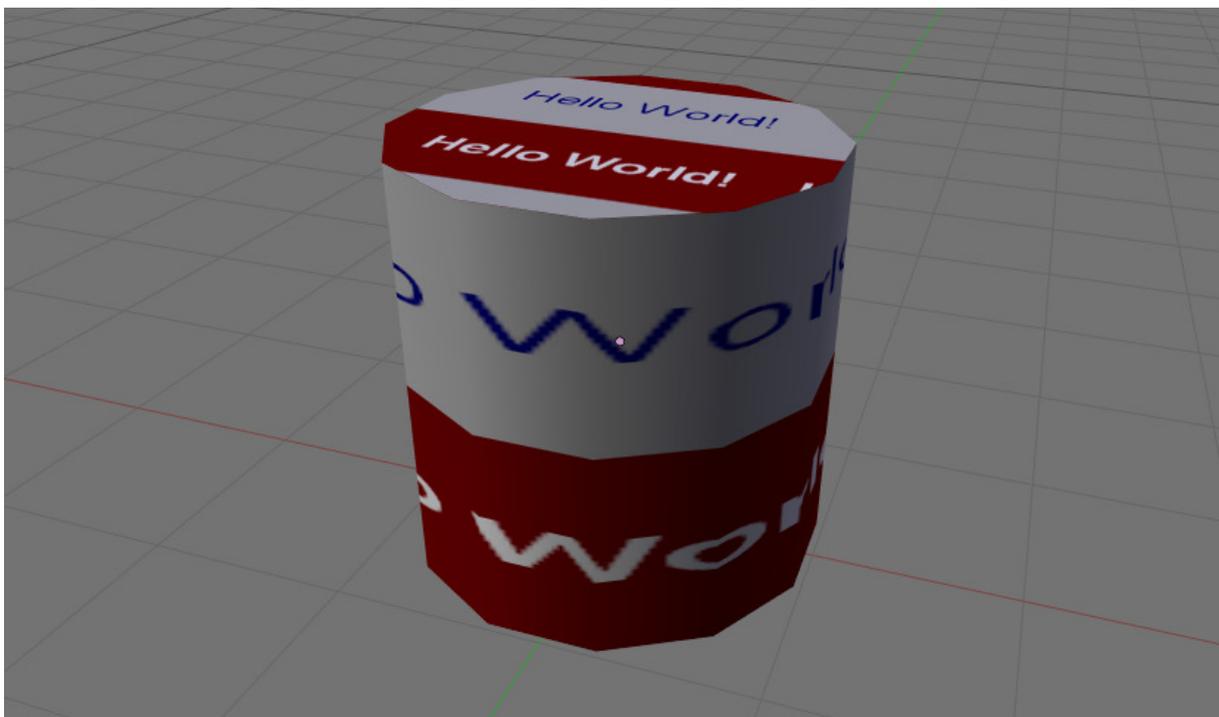


Nun folgt der aufwändigste Schritt: Alle gewollten Kanten müssen durch Trennung der Polygone wiederhergestellt werden. In diesem Beispiel reicht

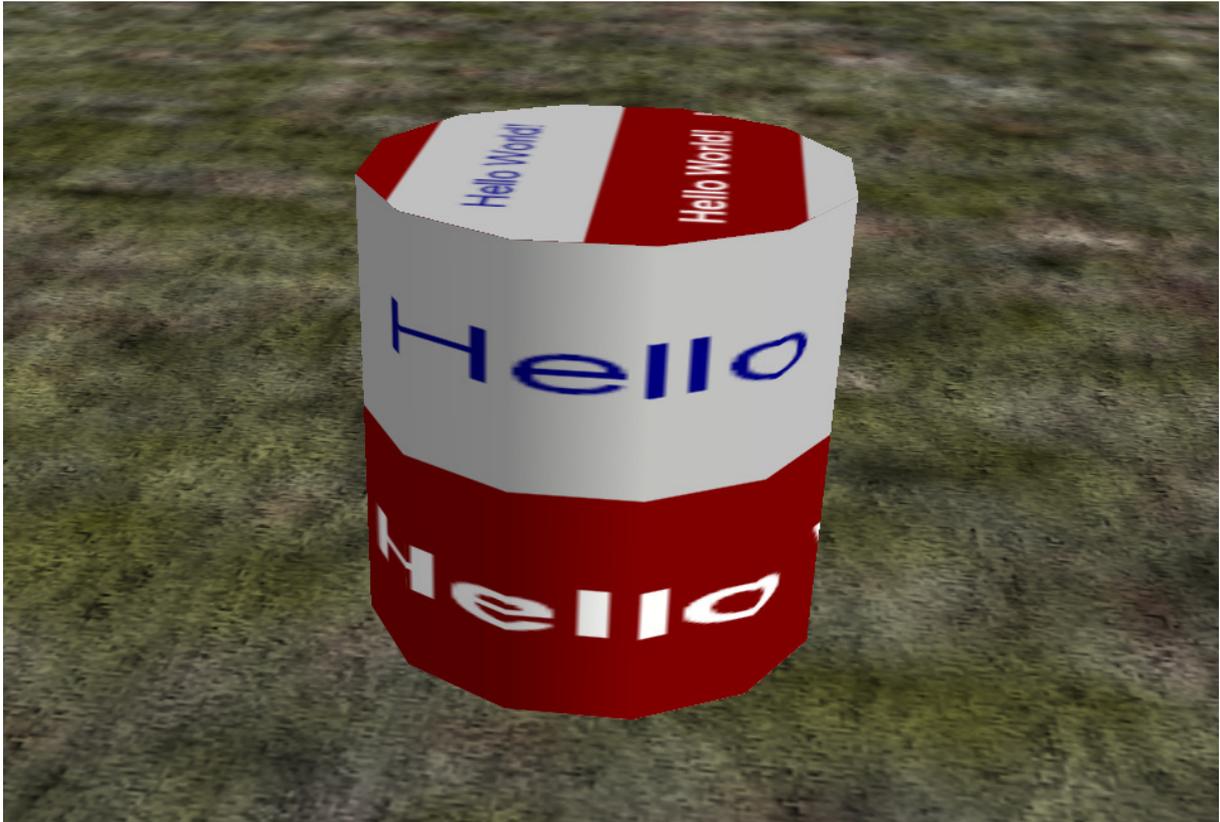
es aus, die Dach- und die Grundfläche des Zylinders (gemeinsam) abzutrennen. Da diese beiden Flächen untereinander nicht miteinander verbunden sind, kann dies hier sogar in einem Schritt erfolgen. Wechseln Sie in den „Edit-Mode“, selektieren Sie alle Polygone, die zur Dach- oder Grundfläche gehören und drücken Sie dann [Y]. Bestätigen Sie dann durch Klick auf „Split“:



Sofort sieht der Zylinder realistisch aus:



Nun noch als dritten Schritt der Export mit „BI.Normals“:



Nun sieht der Zylinder genau richtig aus!

1.9. Physikalische Eigenschaften

Als nächsten wollen wir uns um die Kollisionseigenschaften kümmern.

Zunächst einmal können Sie den aktuellen (Standard-)Zustand testen, indem Sie einfach mal gegen den Zylinder gefahren:



Zunächst wollen wir die Masse und den Schwerpunkt beeinflussen. Fügen Sie die folgenden Befehle der sco-Datei hinzu:

```
[mass]
{Masse in Tonnen}

[momentofintertia]
{Trägheitsmoment x-Achse}
{Trägheitsmoment z-Achse}
{Trägheitsmoment y-Achse}

[cog]
{Schwerpunktposition, x-Koordinate}
{Schwerpunktposition, y-Koordinate}
{Schwerpunktposition, z-Koordinate}
```

Der Befehl [mass] dient der Einstellung der Masse, der Standardwert ist eine Tonne.

Der Befehl [momentofintertia] (da hat sich ein Tippfehler eingeschlichen, der Befehl muss genauso geschrieben werden, auch wenn es „*moment of inertia*“ heißen müsste...) dient der Einstellung der Trägheitsmomente um x-, z- und y-Achse (Reihenfolge beachten!).

Da dieser Wert im Gegensatz zur Masse eher ungeläufig ist, hierzu eine kurze Erklärung:

Das Trägheitsmoment um eine der drei Objektachsen sagt etwas darüber aus, wie träge das Objekt reagiert, wenn ein Drehimpuls auf das Objekt um diese Achse einwirkt. So können zwei Objekte gleicher Masse dennoch unterschiedlich reagieren: Ist die Masse hauptsächlich im Zentrum des Objektes angeordnet, lässt es sich „leicht“ drehen, ist die Masse weit außerhalb des Schwerpunktes angeordnet, reagiert das Objekt wesentlich träger. Die Einheit des Trägheitsmomentes beträgt hier Tonnen * m², eine schwer vorstellbare Einheit. Sie können diesen Wert entweder ausrechnen (siehe hierzu bspw.

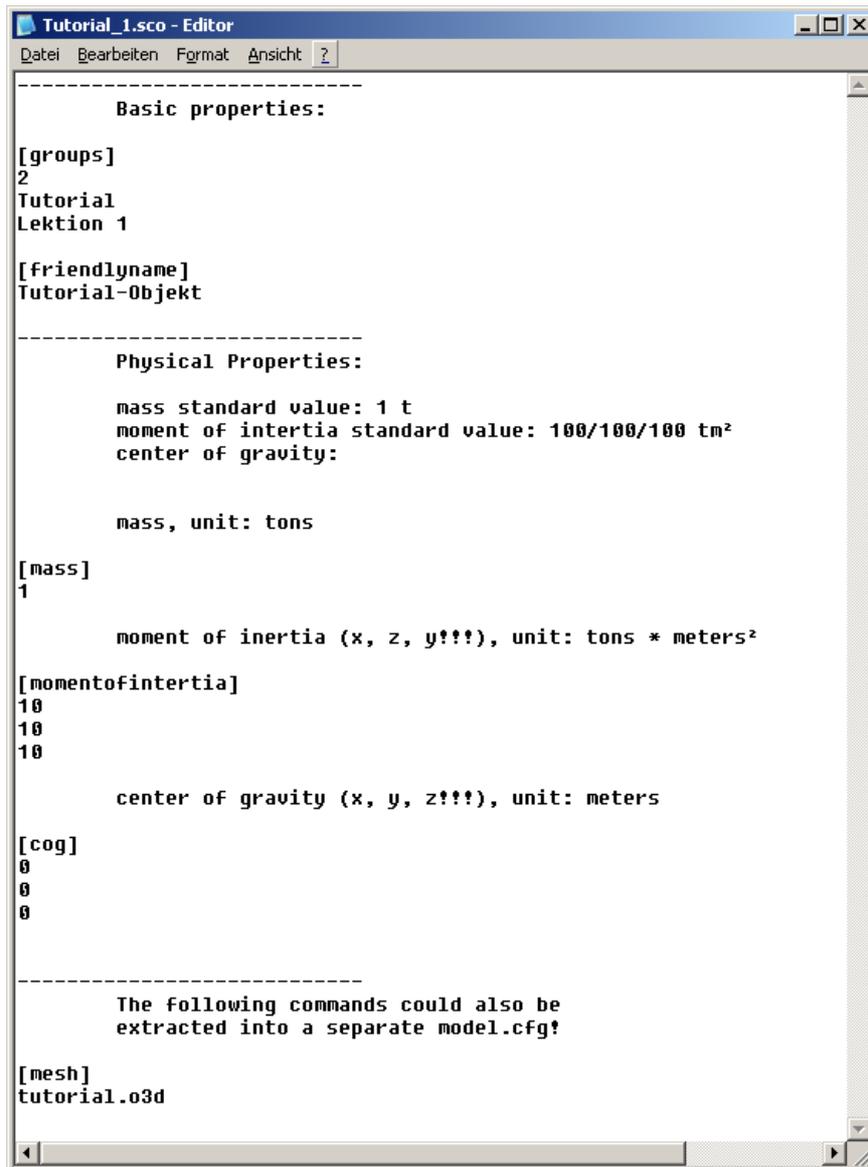
http://de.wikipedia.org/wiki/Tr%C3%A4gheitsmoment#Haupttr.C3.A4gheitsmomente_einfacher_geometrischer_K.C3.B6rper) oder sich schrittweise herantasten und die Werte in OMSI testen.

Der Befehl [cog] schließlich dient der Einstellung des Schwerpunktes, welcher vom Konstruktionsursprung des Objektes abweichen darf. Die Reihenfolge der Achsen ist hier x, y, z.

Wichtig: Die Achsen in OMSI entsprechen den Achsen in Blender (wenn Sie wie beschrieben exportieren); dies bedeutet: x = nach rechts, y = nach vorne, z = nach oben!

Diese Befehle gehören zum Szenerieobjekt, muss also in jedem Fall in der sco-Datei (oder in der ovh-/bus-Datei) stehen.

In unserem Beispiel werden wir nun beginnen, der sco-Datei ein wenig Struktur mit Kommentarzeilen zu geben:



```
Tutorial_1.sco - Editor
Datei Bearbeiten Format Ansicht ?

-----
Basic properties:

[groups]
2
Tutorial
Lektion 1

[friendlyname]
Tutorial-Objekt

-----
Physical Properties:

mass standard value: 1 t
moment of inertia standard value: 100/100/100 tm²
center of gravity:

mass, unit: tons

[mass]
1

moment of inertia (x, z, y!!!), unit: tons * meters²

[momentofintertia]
10
10
10

center of gravity (x, y, z!!!), unit: meters

[cog]
0
0
0

-----
The following commands could also be
extracted into a separate model.cfg!

[mesh]
tutorial.o3d
```

Da der Schwerpunkt normalerweise automatisch so berechnet wird, dass er im Mittelpunkt der ebenfalls automatisch berechneten *Boundingbox* liegt, jetzt aber in den Fußpunkt des Objektes (0,0,0) verschoben wurde, wird sich das Objekt signifikant anders verhalten – probieren Sie es aus!

Interessant ist ebenfalls ein Test mit einer abweichenden Masse (z.B. 0.1) und abweichenden und unterschiedlichen Trägheitsmomenten (z.B. 0.01,1,0.01)! Testen Sie ebenfalls verschiedene Schwerpunktlagen!

Schließlich erlaubt Ihnen OMSI, das Objekt an einem Punkt am Boden zu verankern – fügen Sie hierfür den folgenden Befehl ein und testen Sie ihn in OMSI!

```
[crashmode_pole]
{Parameter 1}
```

```
{Parameter 2}
```

Die Parameter haben zur Zeit keine Bedeutung. Setzen Sie sie auf beliebige Werte, z.B. 1. Weglassen dürfen Sie sie jedoch nicht, damit der Befehl einwandfrei erkannt wird.

1.10. Kollisionseigenschaften

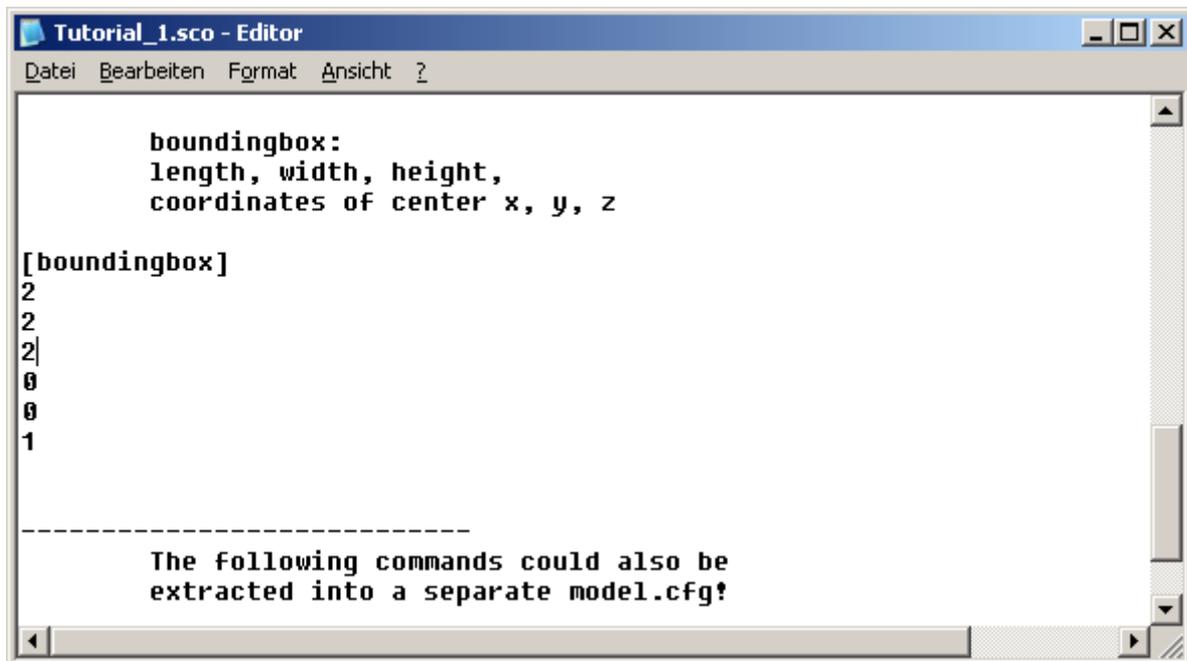
Bei allen Tests werden Sie aber feststellen, dass der Zylinder keineswegs wie ein Zylinder kollidiert – sondern eher wie ein Würfel, in den der Zylinder gerade so hineinpasst.

Dies ist die Standardeinstellung für jedes Szenerieobjekt: Wenn nicht anders angegeben, wird eine sog. *BoundingBox* definiert, ein Quader, in den das Szenerieobjekt gerade so hineinpasst und der entlang der Standardachsen x, y und z ausgerichtet ist.

Diese BoundingBox können Sie ebenfalls manuell einstellen mit folgendem Befehl:

```
[boundingbox]
{Länge (x-Achse)}
{Breite (y-Achse)}
{Höhe (z-Achse)}
{x-Koordinate Mittelpunkt}
{y-Koordinate Mittelpunkt}
{z-Koordinate Mittelpunkt}
```

Die automatische Berechnung liefert folgende Werte:



```
Tutorial_1.sco - Editor
Datei Bearbeiten Format Ansicht ?

    boundingbox:
    length, width, height,
    coordinates of center x, y, z

[boundingbox]
2
2
2|
0
0
1

-----

The following commands could also be
extracted into a separate model.cfg!
```

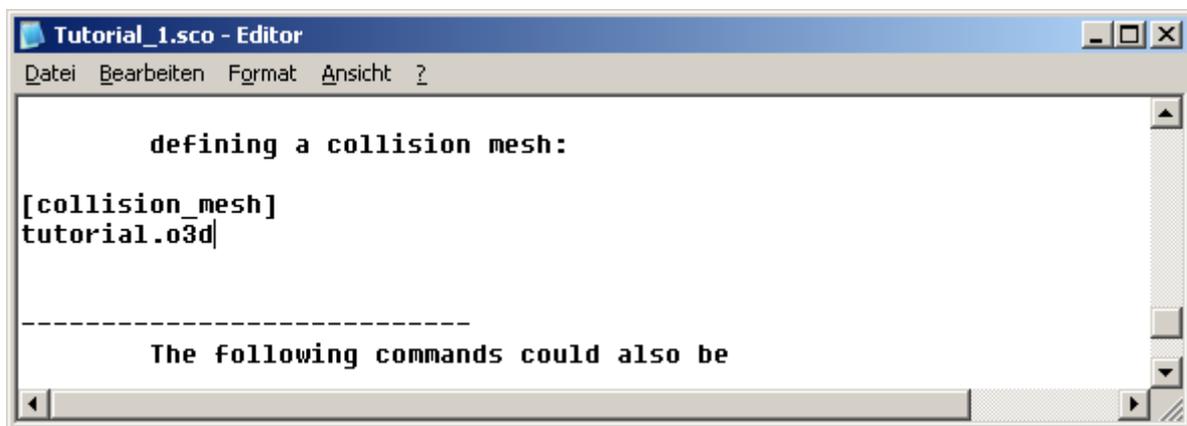
Testen Sie auch hier verschiedene Werte!

Soll nun aber die „Kollisions-Form“ stimmen, müssen Sie dem Objekt ein Kollisions-Mesh zuweisen. Dies *kann* auch das Hauptmesh sein (so werden wir dies hier demonstrieren), muss aber nicht! Das Kollisions-Mesh kann eine eigens exportierte o3d-Datei sein, die bspw. eine sehr viel einfachere Form aufweisen kann als das visuelle Mesh, was den Berechnungsvorgang beschleunigen kann und etwaige Berechnungsfehler ausgleicht, die eventuell aufgrund zu komplexer oder problematischer Formen entstehen können (bspw. wenn das Mesh „offen“ ist).

Probieren Sie also folgenden Befehl aus:

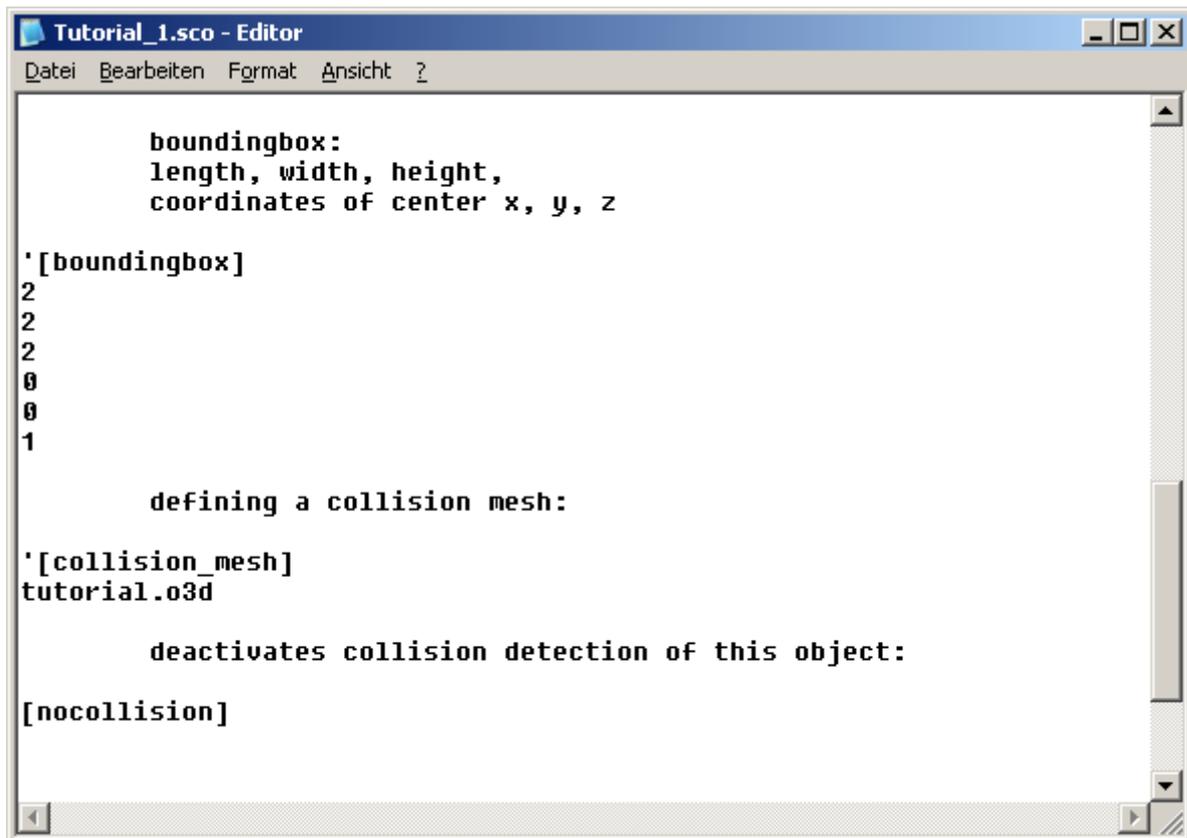
```
[collision_mesh]
{Name der o3d-Meshdatei}
```

Wichtig: Dieser Befehl funktioniert nur mit o3d-Dateien!



Hinweis: Leider gibt es mit Kollisions-Meshs immer wieder Probleme – die Berechnungsvorgänge sind leider recht kompliziert. Auch bei diesem Demonstrationsobjekt kommt es leider dazu, dass das Objekt auch schon mal durch den Boden durchfällt. Probieren Sie einfach verschiedene Objekte aus!

Sie können die Kollisionserkennung auch komplett abschalten, verwenden Sie hierfür den Befehl [nocollision]. Beachten Sie, dass ich den Bounding-Box-Befehl und den Kollisions-Mesh-Befehl im folgenden Beispiel auskommentiert habe, indem ich einen Apostroph davor gesetzt habe. Ich hätte ebenso auch ein Leerzeichen nehmen können oder die Zeile in irgendeiner anderen Form verändern können!



```
boundingbox:
length, width, height,
coordinates of center x, y, z

'[boundingbox]
2
2
2
0
0
1

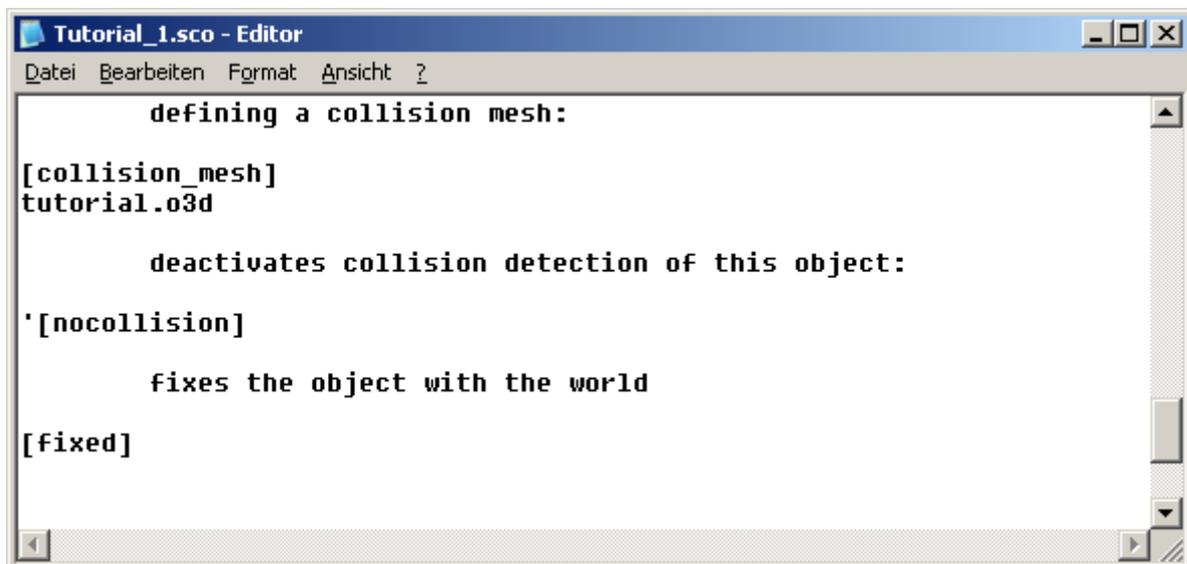
    defining a collision mesh:

'[collision_mesh]
tutorial.o3d

    deactivates collision detection of this object:

[nocollision]
```

Umgekehrt können Sie aber das Objekt auch fixieren! Hierfür fügen Sie den Befehl `[fixed]`; außerdem habe ich das Kollisionsmesh wieder aktiviert:



```
    defining a collision mesh:

[collision_mesh]
tutorial.o3d

    deactivates collision detection of this object:

'[nocollision]

    fixes the object with the world

[fixed]
```

Der typische Anwendungsfall hierfür ist ein Gebäude oder eine Kreuzung, welche natürlich nicht von einem Bus weggeschoben werden können!

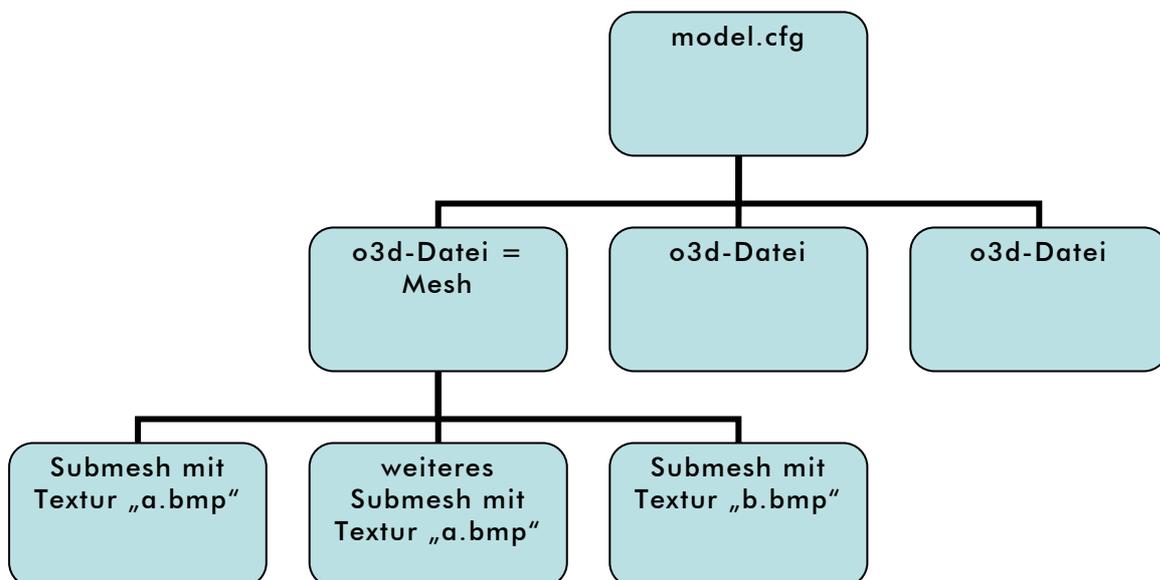
Lektion 2: Materialeigenschaften

Abgesehen von den Beleuchtungsparametern, die direkt in der x-Datei bearbeitet werden, gibt es eine große Anzahl von Materialeigenschaften, die in der model.cfg (oder, falls diese nicht verwendet wird, direkt in der *.sco-, *.ovh- oder *.bus-Datei) eingetragen werden müssen. Diese werden in dieser Lektion vorgestellt.

2.1. Einführung / Theorie

Zunächst etwas Theorie über diese nicht ganz einfache Lektion! Wer möchte, kann natürlich schon einmal vorblättern und experimentieren – sollte aber später hierher zurückfinden, um auch die unverständlich gebliebenen Sachverhalte „nachzuholen“! ;-)

Folgendes Diagramm zeigt zunächst schematisch die Nomenklatur, die nun verwendet wird:



Jede Model.cfg legt mit jedem Befehl [mesh] ein neues Mesh an, welches der aufgerufenen o3d-Datei entspricht.

Jedes Mesh bzw. jede o3d-Datei besteht wiederum aus einer Reihe von Submeshs. Aber jedes Submesh wird nur mit einer Grundtextur belegt!

Für den Export aus Blender gelten üblicherweise folgende Regeln:

- Bestand die o3d-Datei in Blender aus mehreren einzelnen Objekten, so schlägt sich dies auch in den Submeshs nieder: Es wurden dann, obwohl u. U. dieselbe Textur zugewiesen wurde, getrennte Submeshs erzeugt.

- Waren einem exportierten Objekt in Blender verschiedene Texturen zugewiesen, dann wird das Objekt wiederum in mehrere Submeshs zerlegt, jeweils eins für jede Textur.

Daraus folgt nun natürlich, dass es durchaus vorkommen kann, dass mehreren Submeshs dieselbe Textur zugewiesen wurde. Dies wurde im Diagramm ebenfalls dargestellt: Dort gibt es zwei Submeshs mit der Textur „a.bmp“ und ein weiteres mit der Textur „b.bmp“.

Innerhalb der model.cfg wird dieses Konzept nun folgendermaßen umgesetzt:

- [mesh]-Befehl für Mesh 1
 - [matl]-Befehl für Submesh „a.bmp“-0
 - *Materialeigenschaft 1*
 - *Materialeigenschaft 2*
 - [matl]-Befehl für Submesh „b.bmp“-0
 - *Materialeigenschaft 1*
 - *Materialeigenschaft 2*
 - *Materialeigenschaft 3*
- [mesh]-Befehl für Mesh 2
 - *u.U. gar keine Materialeigenschaften*

Wichtig: Die Reihenfolge der [mesh]-Befehle entscheidet über die Renderreihenfolge! Die Reihenfolge der [matl]-Befehle dagegen ist völlig irrelevant, weil die Identifikation über die Textur und eine zusätzliche Indexzahl erfolgt. Auch die Reihenfolge der Materialeigenschaften ist (meistens) ohne Bedeutung.

Zwar können einige Materialeigenschaften, auf die später näher eingegangen wird, über dynamische Eigenschaften verfügen, aber das obige (theoretischen) Beispiel ist dennoch völlig statisch, die Materialeigenschaften bleiben immer konstant.

Dagegen ist es mit einem etwas anderen Aufbau auch möglich, die Materialeigenschaften dynamisch zu verändern! Hierzu wird statt dem Befehl [matl] der Befehl [matl_change] sowie [matl_item] verwendet! Dann folgt der Aufbau folgendem Muster:

- [mesh]

- [matl_change]-Befehl für Submesh „a.bmp“-0 und einer bestimmten Variable
 - Materialeigenschaft 1 für alle Zustände
 - Materialeigenschaft 2 für alle Zustände
 - ...
 - [matl_item]
 - Materialeigenschaft 1 für var = 1
 - Materialeigenschaft 2 für var = 1
 - ...
 - [matl_item]
 - Materialeigenschaft 1 für var = 2
 - ...
 - ggf. weitere [matl_item]-Befehle
- ggf. weitere [matl_change] ODER [matl]-Befehle
 - ggf. weitere [mesh]-Befehle

Es wird hier also je nach Zustand der (ganzzahlig gerundeten) Variable entschieden, welche Materialeigenschaften verwendet werden!

Zur Übersicht hier eine Liste sämtlicher Material-Befehle, die von OMSI 1.01 unterstützt werden:

[alphascale]	Alphakanal wird mit Variable multipliziert
[matl_allcolor]	Neusetzen der Materialfarben
[matl_alpha]	Darstellung von Transparenz
[matl_bumpmap]	Bumpmap (nur in Verbindung mit Envmap)
[matl_envmap]	Envmap („Reflexionstextur“)
[matl_envmap_mask]	Envmap, abhängig vom Alphakanal
[matl_freetex]	Material verwendet „Freetexture“
[matl_lightmap]	Fügt Lightmap hinzu
[matl_nightmap]	Nightmap
[matl_noZwrite]	Material schreibt nicht in den Z-Buffer
[matl_texaddress_border]	Texturadressierung im „Border“-Modus
[matl_texaddress_clamp]	Texturadressierung im „Clamp“-Modus
[matl_texaddress_mirror]	Texturadressierung im „Mirror“-Modus
[matl_texaddress_mirroronce]	Texturadressierung im „Mirror-Once“-Modus
[matl_transmap]	Transparenzmap

<code>[texcoordtransY]</code>	Verschieben der Y-Texturkoordinate
<code>[useTexture]</code>	Material verwendet Text-Textur

2.2. **`[matl]`, `[matl_change]` und `[matl_allcolor]`**

Im ersten Beispiel stellen wir die Befehle `[matl]`, `[matl_change]` und `[matl_allcolor]` vor.

Wir haben für die folgenden Tests das Objekt `Tutorial_2.sco` erstellt. Es enthält ein Script mit zwei Variablen, welche für die Demonstrationen von Material-Animationen ständig zwischen 0 und 3 wechseln, wobei die Variable „discrete“ dabei nur die ganzzahligen Werte 0, 1 und 2 einnimmt, die Variable „float“ aber alle Werte von 0 bis 2,999.

Kopieren Sie den Ordner „Tutorial_2“ in Ihren OMSI ins „\Sceneryobjects“-Verzeichnis.

Öffnen Sie die `tutorial_2.sco`-Datei. Sie entspricht in etwa dem Stand am Ende der Lektion 1, auch das `o3d`-Mesh ist das gleiche.

Der letzte Eintrag ist der folgende:

```
[mesh]
tutorial.o3d
```

Wir wollen nun mit Hilfe des Befehls `[matl_allcolor]` den Emissive-Anteil des Materials auf die Farbe Rot setzen, sodass das Objekt rot leuchtet. Hierzu wird zunächst das Submesh mit Hilfe des Texturnamens herausgesucht:

```
[matl]
sdk1.bmp
0
```

Der Befehl hat folgende Syntax:

```
[matl]
Texturname
Zusatzindex
```

Beachten Sie: Sollte die `o3d`-Datei über mehrere Submeshs mit derselben Textur verfügen (wenn die exportierte Blenderdatei über mehrere Objekte mit derselben Textur verfügt, kann dies passieren!), dann können Sie mit dem Zusatzindex (hier „0“) entscheiden, für welches Submesh die nun folgenden Befehle gelten!

Mit diesem Befehl haben wir nun also OMSI mitgeteilt, dass wir gerne die Materialeigenschaften vom Submesh mit der Textur „`sdk1.bmp`“ verändern wollen!

Als nächstes folgt der Befehl [matl_allcolor], mit dem wir dieselben Farbwerte manipulieren können, die wir auch schon in 1.7 direkt in der x-Datei verändert haben:

```
[matl_allcolor]
1
1
1
1
1
1
1
1
1
0
0
0
1
0
0
0
```

Die Syntax dieses Befehl lautet:

```
[matl_allcolor]
Diffuse (red)
Diffuse (green)
Diffuse (blue)
Diffuse (alpha)
Ambient (red)
Ambient (green)
Ambient (blue)
Specular (red)
Specular (green)
Specular (blue)
Emissive (red)
Emissive (green)
Emissive (blue)
Power
```

Entsprechend ähnlich wie vorhin auch das Ergebnis:



Dieses Ergebnis ist nun wenig spektakulär, weil dasselbe ja bereits mit dem Wissen aus Lektion 1 erreicht werden kann. Interessant wird es nun aber, weil wir jetzt den Befehl [matl_change] verwenden!

Ändern Sie den Befehl [matl] um in [matl_change] und ergänzen Sie als dritten Parameter „discrete“. Fügen Sie außerdem zwischen diesem Befehl

und dem [matl_allcolor]-Befehl den Befehl [matl_item] hinzu, sodass alles zusammen folgendermaßen aussieht:

```
[mesh]
tutorial.o3d

[matl_change]
sdk1.bmp
0
discrete

[matl_item]

[matl_allcolor]
1
1
1
1
1
1
1
1
0
0
0
1
0
0
0
```

Die Variable „discrete“ wechselt sekundlich den Wert von 0 auf 1 und 2 und dann wieder auf 0.

Der Befehl [matl_item] arbeitet ähnlich wie eine select/case-Anweisung in C/Pascal: Zunächst werden alle Materialeigenschaften gesetzt, die vor dem ersten [matl_item]-Befehl kommen. Dann wird die Variable geprüft: Ist sie 1, dann werden alle Materialeigenschaften verwendet, die nach dem ersten [matl_item]-Eintrag folgen, ist sie 2, dann die nach dem zweiten usw.

Wenn Sie das Objekt testen, werden Sie feststellen, dass das Objekt „blinkt“: 2 Sekunden lang ist es dunkel, 1 Sekunde leuchtet es rot.

Wir können das ganze noch um einen weiteren Eintrag ergänzen:

```
[matl_change]
sdk1.bmp
0
discrete

[matl_item]

[matl_allcolor]
1
1
1
1
1
```

```

1
1
0
0
0
1
0
0
0
0

[matl_item]

[matl_allcolor]
1
1
1
1
1
1
1
1
1
0
0
0
0
0
0
1
0

```

Nun blinkt das Objekt stets in der Reihenfolge „dunkel“ => „rot“ => „blau“ im Sekundentakt.

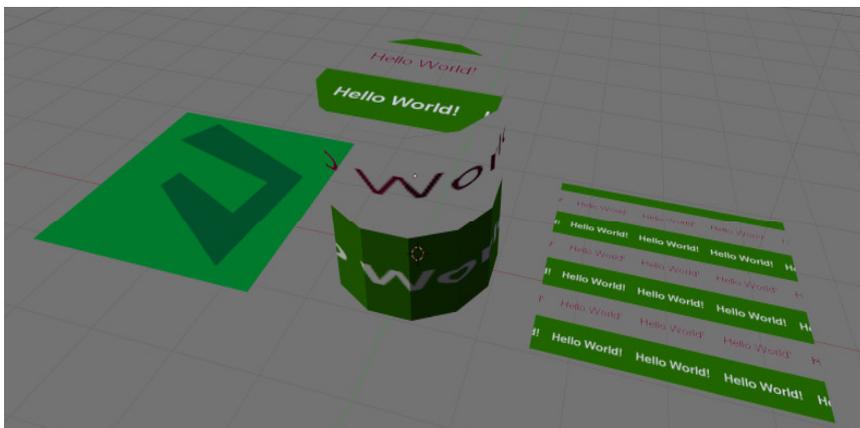
Für die folgenden weiteren Experimente werden wir auf ein etwas komplexeres Objekt zurückgreifen. Ändern Sie daher den [mesh]-Eintrag wie folgt:

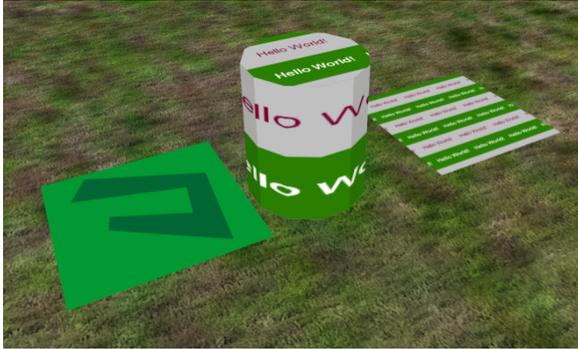
```

[mesh]
sdk2_2_1.o3d

```

Dieses Objekt besteht aus drei Objekten, von denen zwei dieselbe Textur haben (sdk3.tga) und das dritte eine andere Textur (sdk2.bmp). Außerdem enthält sdk3.tga einen Alpha-Kanal für weitere Experimente.





Blender hat den Alphakanal bereits als Transparenzkanal dargestellt. Dies muss OMSI aber gesondert mitgeteilt werden... dazu aber später mehr.

Zunächst soll hier das Prinzip der [matl]-Befehle noch einmal vertieft werden: Dadurch, dass es zwei Objekte (Submeshs) mit derselben Textur gibt (sdk3.tga) muss nun vom Zusatzindex gebrauch gemacht werden. Der „Turm“ hat dabei den Index 0, die „Fläche“ den Index 1. Wir wollen nun den Turm rot, die Fläche blau und die vordere Fläche mit der anderen Textur grün leuchten lassen.

Es sind also folgende Kommandos nötig:

```
[matl]
sdk3.tga
0
```

```
[matl_allcolor]
1
1
1
1
1
1
1
1
1
0
0
0
0
1
0
0
0
```

```
[matl]
sdk3.tga
1
```

```
[matl_allcolor]
1
1
1
1
1
1
1
1
0
```

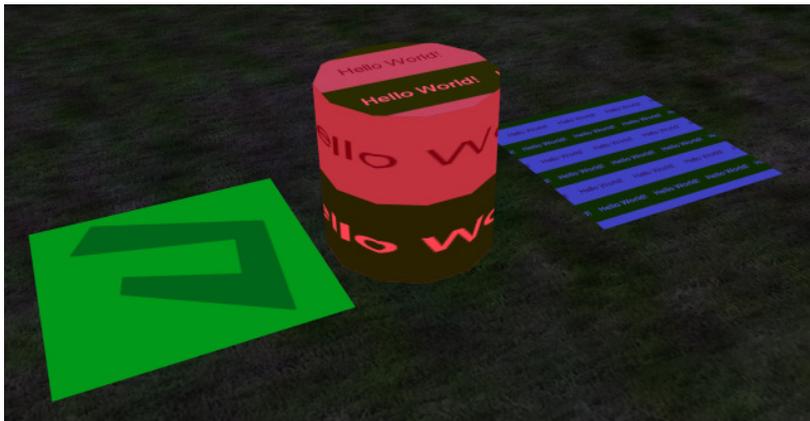
```

0
0
0
0
1
0

[matl]
sdk2.bmp
0

[matl_allcolor]
1
1
1
1
1
1
1
1
1
1
0
0
0
0
1
0
0

```



Zuletzt soll darauf eingegangen werden, was passiert, wenn man einen [matl]-Eintrag falsch vornimmt. Was passiert, wenn man z.B. das Submesh sdk2.bmp-1 mit Materialeigenschaften versehen will, welches nicht vorhanden ist?

Wenn wir beim letzten [matl]-Befehl die 0 in eine 1 ändern, wird zunächst die vordere Fläche natürlich nicht mehr grün leuchten. Außerdem gibt es folgenden Eintrag in der Log-Datei (logfile.txt):

```

6 16:34:13 - - Warning: File
Sceneryobjects\Tutorial\Tutorial_2.sco: texture filename sdk2.bmp
not found in mesh file Sceneryobjects\Tutorial\model\sdk2_2_1.o3d!

```

Diese Meldung ist etwas missverständlich, da sie davon ausgeht, dass der Texturname nicht stimmt – tatsächlich aber stimmt der Index nicht... dennoch ist die Log-Datei beim Zuweisen von Materialeigenschaften

gelegentlich zu prüfen, insbesondere aber dann, wenn etwas nicht so läuft, wie man es sich vorstellt!

2.3. **Nightmap, Lightmap, Envir-Map, Bumpmap**

OMSI unterstützt sowohl Nachttexturen (leuchten „unabhängig“ von der Originaltextur), Lichttexturen (werden mit Originaltextur multipliziert), Envir-Maps (Reflexionen) und auch Bumpmaps (unregelmäßige Oberflächen in Verbindung mit Reflexionstexturen).

Wir testen zunächst die Nachttextur. Beachten Sie: „Nachttextur“ bedeutet hierbei NICHT, dass sie nur nachts aktiv ist! Es ist eine reine Beschreibung für die grafische Verwendung der Textur.

Syntax:

```
[mesh]
sdk2_2_1.o3d
```

```
[mat1]
sdk3.tga
0
```

```
[mat1_nightmap]
nightmap.bmp
```

Geben Sie lediglich den Dateinamen der zu verwendenden Nachttextur an.

Eine Nachttextur wird grundsätzlich *nach* der Berechnung der Haupttextur hinzugefügt, dies unterscheidet sie von der Lichttextur:



Eine andere Variante zur Beleuchtung mit Hilfe von Texturen stellen die Lightmaps / Lichttexturen dar. Hierbei wird zuerst die Lichttextur auf das Objekt aufgebracht und dann erst die Haupttextur. Der Vorteil ist, dass die Auflösung auch wesentlich geringer sein kann (wie in diesem Beispiel), weil die Details weiterhin von der Haupttextur verwendet werden; außerdem können mehrere Lichttexturen überlagert werden, was bspw. bei der Innenbeleuchtung (Oberdeck und Unterdeck) der SD200 und SD202 genutzt wird.

Nachteil ist aber, dass die Lichttextur von der Helligkeit her immer von der Haupttextur begrenzt wird. Dort schwarze Flächen können niemals von der Lichttextur „aufgehellt“ werden.

Aufgrund der Mehrfachnutzung enthält der Befehl auch optional einen Dateinamen:

```
[mesh]
sdk2_2_1.o3d

[mat1]
sdk3.tga
0

[mat1_lightmap]
lightmap.bmp
discrete
```

In diesem Fall wurde die Variable „discrete“ mit der Lichttextur „lightmap.bmp“ verbunden. Sie könnten nun eine weitere Lichttextur mit einer anderen Variable hinzufügen, sodass je nach Variablenwert eine der beiden oder sogar beide gleichzeitig aktiv sind.

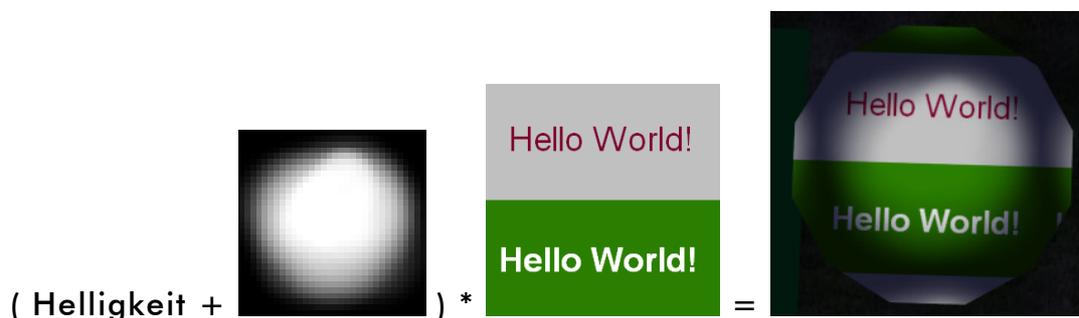
Wenn die Lichttextur immer aktiv sein soll, setzen sie einfach einen ungültigen Variablennamen ein oder lassen Sie die Zeile leer:

```
[mesh]
sdk2_2_1.o3d

[mat1]
sdk3.tga
0

[mat1_lightmap]
lightmap.bmp
```

Das Ergebnis sieht dann so aus:



Je nachdem, ob Sie nun die Variable verwenden oder nicht, ist die Lichttextur entweder immer aktiv oder nur, wenn die Variable ungleich 0 ist – in diesem Fall also zwei Sekunden an (discrete = 1 oder 2) und eine Sekunde aus (discrete = 0).

Kommen wir nun zu den Reflexionstexturen. Hierbei handelt es sich um eine mehr oder weniger gut geeignete, statische Textur, die in etwa die Umgebung des Objektes widerspiegelt. Im Gegensatz zu rechenintensiven Echtzeittexturen handelt es sich also um eine Behelfslösung, die aber in den meisten Fällen völlig ausreicht, da das Auge ohnehin meistens nicht genau wahrnimmt, was sich spiegelt, sondern nur, dass sich etwas spiegelt.

Die Syntax ist wiederum nicht kompliziert: Es wird lediglich der Texturname angegeben sowie die Intensität, mit der die Reflexion überlagert wird:

```
[mesh]
sdk2_2_1.o3d
```

```
[mat1]
sdk3.tga
0
```

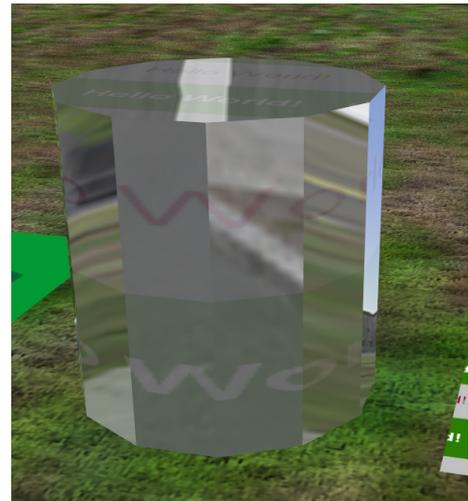
```
[mat1_envmap]
envmap.bmp
0.5
```



Je nach Wahl der Intensität erhalten Sie einen Effekt wie auf einer Metalloberfläche oder sogar einen Chromeffekt:



0.1:



0.9:

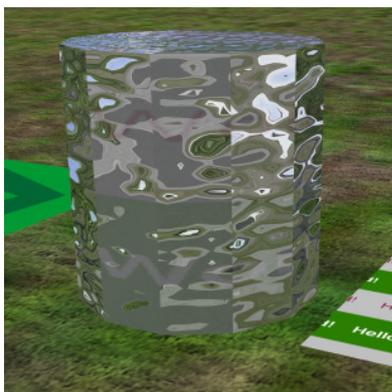
Schließlich kann die reflektierende Oberfläche noch um eine Bumpmap ergänzt werden, welche zusätzlich Oberflächenstrukturen erscheinen lässt. Auch hier gibt es wieder einen Intensitätsfaktor, der darüber entscheidet, wie intensiv das Bumpmapping ausgeführt wird.

```
[mesh]
sdk2_2_1.o3d

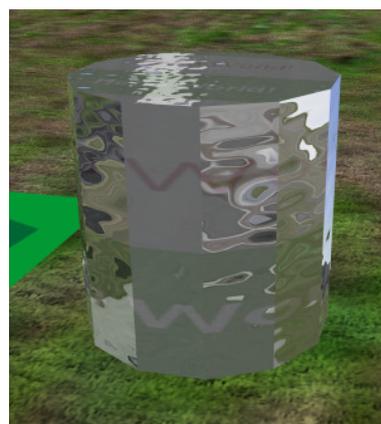
[mat1]
sdk3.tga
0

[mat1_envmap]
envmap.bmp
0.9

[mat1_bumpmap]
bumpmap.bmp
0.5
```



, mit Faktor = 0.1:



Wichtig: Zum Schluss soll noch mal darauf hingewiesen werden, dass nicht etwa für jeden [matl_...]-Befehl ein eigener [matl]-Befehl verwendet wird! Verwenden Sie stets pro Submesh nur einen [matl] oder [matl_change] Befehl als „Einleitung“, gefolgt von der Reihe von anzuwendenden Materialeigenschaften.

Sollen also alle vier hier vorgestellten Maps gleichzeitig angewendet werden, müssen die Befehle folgendermaßen angewendet werden (Reihenfolge ist aber nicht relevant):

```
[mesh]
sdk2_2_1.o3d

[matl]
sdk3.tga
0

[matl_nightmap]
nightmap.bmp

[matl_lightmap]
lightmap.bmp
discrete

[matl_envmap]
envmap.bmp
0.3

[matl_bumpmap]
bumpmap.bmp
0.1
```

2.4. Verwendung des Alphakanals als Transparenz

Wenn die Textur einen Alphakanal enthält (bspw. beim Dateiformat Targa), kann dieser von OMSI auf verschiedene Weise genutzt werden. Die einfachste und bekannteste Form ist als direkte Nutzung als Transparenzkanal.

Hierbei gibt es zwei unterschiedliche Varianten:

- **Typ 1: Scharfes „Ausschneiden“.** Hierbei entstehen harte Kanten, die jedoch nur in bestimmten Fällen gewünscht werden (Zäune, eventuell Pflanzen) und es sind keine Teiltransparenzen möglich. Der Vorteil ist aber, dass es zu keinen Problemen mit der Renderreihenfolge kommen kann.
- **Typ 2: Weiche Überblendung.** Die Gesamtdarstellung ist weicher (was manchmal aber auch unerwünscht sein kann), Teiltransparenzen sind möglich, aber es kommt schnell zu Problemen mit der Renderreihenfolge.

Testen wir beide Varianten. Der Befehl ist lediglich [matl_alpha], gefolgt vom Typ:

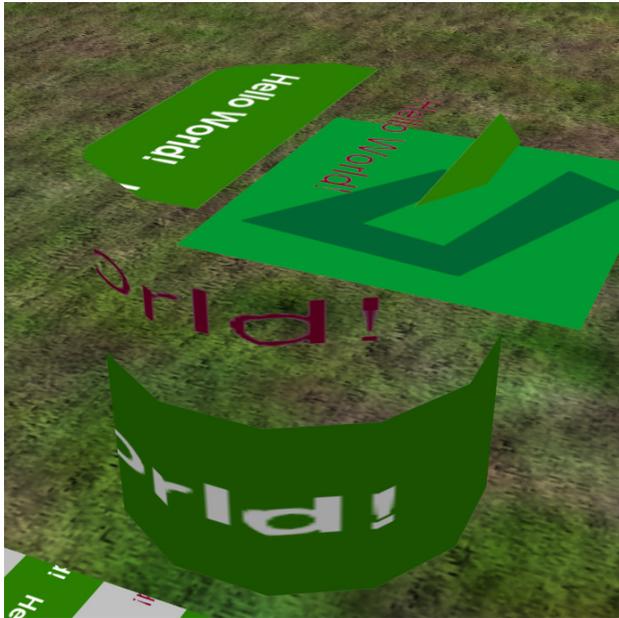
```
[mesh]
sdk2_2_1.o3d

[matl]
sdk3.tga
```

0

```
[mat1_alpha]
```

1

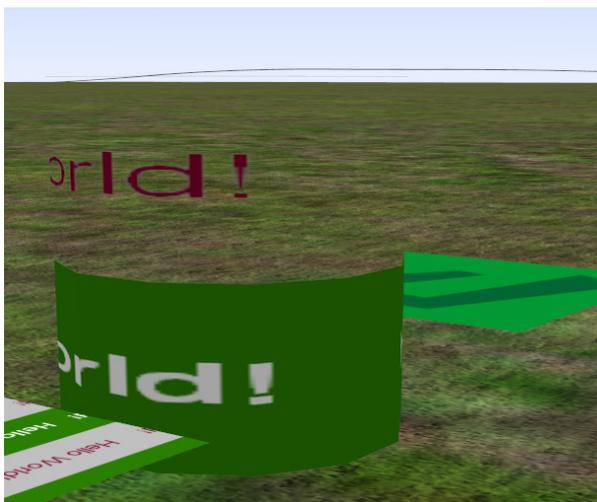


Bei Verwendung des Typs 2 treten sofort die Probleme mit der Renderreihenfolge auf:

```
[mesh]  
sdk2_2_1.o3d
```

```
[mat1]  
sdk3.tga  
0
```

```
[mat1_alpha]  
2
```



Das Problem hierbei ist, dass zuerst die Säule gerendert wird und erst danach die grüne Fläche dahinter. Da der Tiefen-Puffer (Funktionsweise:

<http://de.wikipedia.org/wiki/Z-Buffer>) jedoch nur binär funktioniert, geht OMSI davon aus, dass die gesamte Säule im Vordergrund die Fläche verdeckt.

Gelöst werden kann das Problem nur durch eine Reihe von Tricks, insbesondere aber indem die Renderreihenfolge sinnvoll gewählt wird. In diesem Fall würde man die beiden Flächen in eine separate o3d-Datei auslagern und als zweiten [mesh]-Befehl aufrufen, der dann nach dem mit der Säule auftaucht. Alternativ kann man in Blender mit Hilfe einiger Tricks die Reihenfolge der Objekte solange manipulieren, bis sie das gewünschte Ergebnis bringt.

Außerdem gibt es den Befehl [rendertype] für *.sco-Dateien, mit denen gezielt angegeben werden kann, wann das Szenerieobjekt gerendert werden soll:

```
[rendertype]
{type}
```

{type} kann dabei folgende Werte annehmen:

- *presurface*: bevor das Terrain und die Oberflächenobjekte gerendert werden (z.B. um „optische Löcher“ in den Boden zu schneiden wie bei den U-Bahneingängen)
- *surface*: zusammen mit dem Terrain und den Straßen
- *on_surface*: unmittelbar nach den Straßen (z.B. Richtungspfeile)
- *1*: werden vor den „normalen“ (nicht gekennzeichneten) Objekten gerendert
- *2*: Standardwert (wenn nicht anders angegeben)
- *3*: werden nach den „normalen“ Objekten gerendert
- *4*: werden nach den Straßenfahrzeugen gerendert, sinnvoll bspw. bei Objekten mit besonders viel Glas wie Haltestellenhäuschen

2.5. Weitere Möglichkeiten der Alphakanalnutzung

Mit Hilfe des Befehls [alphascale] kann der Alphakanal mit einer beliebigen Variable multipliziert werden:

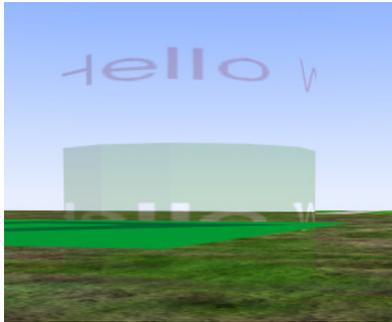
```
[mesh]
sdk2_2_1.o3d
```

```
[mat1]
sdk3.tga
0
```

```
[alphascale]
```

```
floating
```

```
[mat1_alpha]  
2
```



Als weitere Möglichkeit kann der Alphakanal genutzt werden, um die Intensität einer Reflexionstextur zu manipulieren. Hierzu wird der normale [matl_envmap]-Befehl um den Befehl [matl_envmap_mask] erweitert:

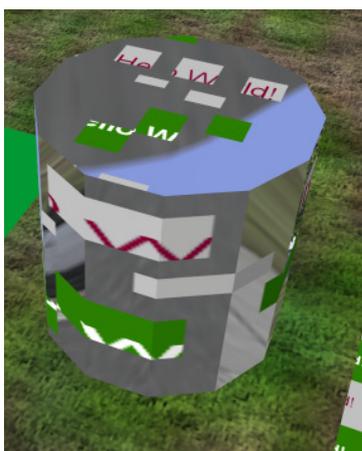
```
[mesh]  
sdk2_2_1.o3d
```

```
[mat1]  
sdk3.tga  
0
```

```
[mat1_envmap]  
envmap.bmp  
0.0
```

```
[mat1_envmap_mask]  
alphamap.tga
```

Relevant bei der Textur „alphamap.tga“ ist lediglich der Alphakanal; es kann sogar eine reine Alphakanal-Textur sein (im *.dds-Format möglich).



In dieser Anwendung wird der beim Befehl [matl_envmap] eingegebene Faktor ignoriert.

Da in diesem Fall keine (vom gewählten Alphakanal abweichende) Transparenz gewählt werden kann, muss im Falle der gleichzeitigen

Notwendigkeit eines Transparenz- und eines Reflexionsintensitätskanals auf den Befehl [matl_transmap] zurückgegriffen werden:

```
[mesh]
sdk2_2_1.o3d

[matl]
sdk3.tga
0

[matl_envmap]
envmap.bmp
0.2

[matl_transmap]
alphamap.tga

[alphascale]
floating

[matl_alpha]
2
```

In diesem Fall verwendet also die Reflexionsintensität den Alphakanal der Haupttextur (diese kann dann auch mit [alphascale] noch variiert werden), die Transparenz wird über die unter [matl_transmap] angegebene Transparenztextur (reine Alphakanal-Textur) definiert:



Die Transparenztextur kann auch ohne Envmap verwendet werden, z.B. wenn die Haupttextur keine Transparenz hat. Allerdings ist der Speicher- und Rendraufwand im Allgemeinen höher, weshalb es grundsätzlich vorzuziehen ist, den Alphakanal zu verwenden, der in der Haupttextur enthalten ist.

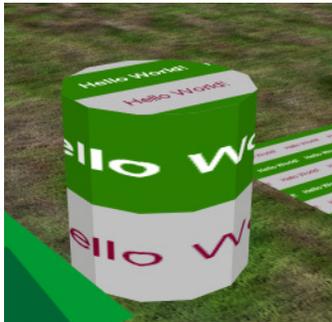
2.6. Animation der Texturkoordinaten

Bspw. für das Rollband kann die y-Koordinate der Texturkoordinaten an eine Variable gebunden werden:

```
[mesh]
sdk2_2_1.o3d

[mat1]
sdk3.tga
0

[texturecoordtransY]
floating
```



2.7. Texturadressmodi

Ein Blick auf das hintere Rechteck zeigt, dass die Textur im Normalfall bei Bedarf sooft wiederholt wird, bis die gesamte Fläche gefüllt ist. Dieser Modus kann geändert werden, hierzu dienen die Befehle `[matl_texaddress_border]`, `[matl_texaddress_clamp]`, `[matl_texaddress_mirror]` und `[matl_texaddress_mirroronce]`:

```
[mesh]
sdk2_2_1.o3d

[mat1]
sdk3.tga
0

[matl_texaddress_border]
255
0
0
0
```



Das Prinzip an sich ist simpel: Außerhalb der eigentlichen Texturkoordinaten wird eine einfache Farbe gewählt, welche durch die

Parameter unter dem Befehl definiert wird. Allerdings sind gleich mehrere Probleme offensichtlich:

Erstens stimmt die Reihenfolge der RGBA-Werte nicht – vielmehr ist die Reihenfolge Blau – Grün – Rot – Alphakanal.

Zweitens entspricht die „Ausgangsposition“ der Textur nicht genau der in Blender – die Originalposition ist um eine Texturlänge nach unten verschoben. Soll dieses System also gezielt eingesetzt werden, ist ein gegensinniges Verschieben nötig.

Drittens hat OMSI 1.01 noch Probleme mit dem Alphakanal (wird in zukünftigen Versionen behoben sein). Hier kann nur 0 gewählt werden. Entweder also der Alphakanal wird nicht genutzt, dann kann die Farbe nach Belieben festgelegt werden oder aber der Alphakanal wird als Transparenz verwendet, dann ist die gesamte Fläche außerhalb der Textur allerdings auch volltransparent.

Die anderen Modi sind einfacher und kommen ohne Parameter aus:

```
[matl_texaddress_clamp]
```



```
[matl_texaddress_mirror]
```



```
[matl_texaddress_mirroronce]
```



Die obigen Beispiele sollten selbsterklärend sein...! ;-)

2.8. Kein Schreiben in Z-Buffer

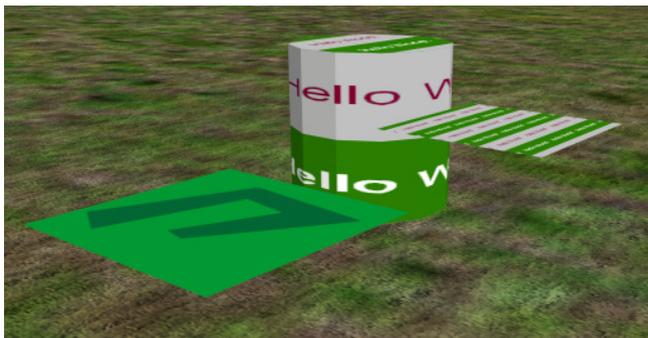
Für die korrekte Darstellung der gerenderten Objekte wird der sogenannte Tiefenpuffer oder auch Z-Buffer verwendet. (Genauere Infos => <http://de.wikipedia.org/wiki/Z-Buffer>)

In bestimmten Situationen kann es sinnvoll sein, das Schreiben in den Z-Buffer zu verhindern, sodass weiter weg liegende Objekte, die danach gerendert werden, dennoch sichtbar sind:

```
[mesh]
sdk2_2_1.o3d

[mat1]
sdk3.tga
0

[mat1_noZwrite]
```



2.9. FreeTex

Hiermit ist ein Verfahren gemeint, bei dem das Script direkt den Dateinamen vorgibt.

Das Script im Tutorialobjekt verfügt über eine Stringvariable „freetex“, welche im Sekundentakt auf die Werte „tex_0.bmp“, „tex_1.bmp“ und „tex_2.bmp“ gesetzt wird. Entsprechende Texturen sind im

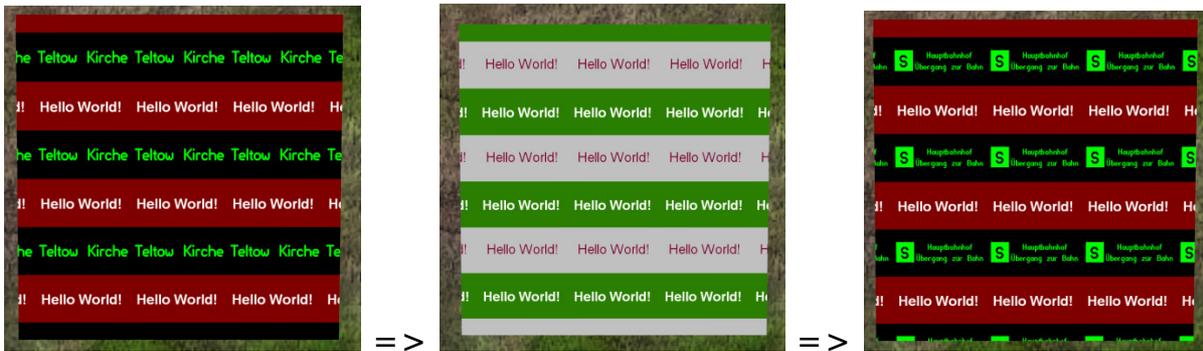
Texturverzeichnis bereits vorhanden. Beachten Sie aber auch, dass die Textur mit der Nummer 1 nicht vorhanden ist!

```
[mesh]
sdk2_2_1.o3d

[matl]
sdk3.tga
1

[matl_freetex]
sdk3.tga
freetex
```

Der erste Parameter ist die Textur, die verändert werden soll, der zweite Parameter die String-Variable, welche den Dateinamen enthält. Auf diese Weise können bspw. Vollmatrix-Zielschilder auf sehr einfache Weise erstellt werden – aber auch Szenerieobjekte so vorbereitet werden, dass im Editor eine von vielen Tauschtexturen als Parameter eingetragen werden kann.



Besonders zu beachten ist bei diesem Befehl, dass durch die Angabe der Originaltextur nicht unbedingt nur die Haupttextur betroffen sein muss! Mit folgenden Befehlen z.B. wird gezielt die Nachttextur getauscht (gerade bei LED-Anzeigen wichtig!):

```
[mesh]
sdk2_2_1.o3d

[matl]
sdk3.tga
1

[matl_nightmap]
nightmap.bmp

[matl_freetex]
nightmap.bmp
freetex
```



2.10. Texttexturen

Texttexturen erlauben es, Stringvariablen in Texturen umzuwandeln, bspw. für die Darstellung von Displayanzeigen (Haltestellenanzeige, IBIS, Zielschild) aber auch von Beschriftungen, die im Editor vorgenommen werden (Haltestellen, Straßenschilder usw.).

Am Anfang werden zunächst die Texttexturen vorbereitet (die dann durchaus mehrfach verwendet werden können):

```
[texttexture]
freetex
IBIS-2_5x7
128
128
0
128
128
255
```

Zunächst wird die Variable angegeben. Wir verwenden wieder „freetex“. Dann folgt der Name des Fonts. Diese Namen können in den jeweiligen *.oft-Dateien nachgelesen werden: Es ist der jeweils erste Parameter im [newfont]-Befehl!

Als nächstes wird die Auflösung der zu erstellenden Textur festgelegt, hier also 128x128 Pixel. Dann folgt eine 0, wenn die Farbe des Fonts manuell festgelegt werden soll oder eine 1, wenn die Originalfarben des Fonts übernommen werden sollen (bspw. für die bunten Haltestellensymbole für U und S-Bahn). Schließlich kommen die drei Farbwerte für den Fall, dass der Modus „0“ genutzt wird, hier hellblau. Wenn „1“ verwendet wird, bitte trotzdem zur Sicherheit drei Nullen eintragen!

Schließlich muss die Texttextur nur noch eingebunden werden mit dem Befehl [useTextTexture] und dem Index 0 (ist ja der „nullte“ [texttexture]-Befehl, den wir hier nutzen wollen!). Außerdem muss [matl_alpha] auf 1 oder 2 gesetzt werden, damit das Ergebnis auch sichtbar wird – denn üblicherweise sind die Fonts dafür ausgelegt, dass der Untergrund durch

ein separates Polygon dargestellt wird und die Schrift per Alphakanal ausgeschnitten wird.

```
[texttexture]
freetex
IBIS-2_5x7
128
128
0
128
128
255

.....

[mesh]
sdk2_2_1.o3d

[mat1]
sdk3.tga
0

[useTextTexture]
0

[mat1_alpha]
1
```

Das sieht dann so aus:



Hinweis fürs Script: Wenn die Texttextur animiert sein soll, muss das Script die Variable „Refresh_Strings“ jedes Mal dann auf 1 setzen, wenn sich der Text verändert hat. Auf diese Weise wird verhindert, dass OMSI ständig prüfen muss, ob sich die entsprechenden Variablen verändert haben.

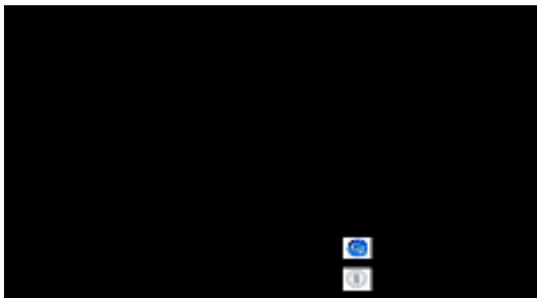
2.11. Eigene Fonts

Zum Abschluss dieser Lektion werfen wir einen Blick auf die Font-Datei „Kennz_DtAlt.oft“ im Verzeichnis „OMSI\Fonts“. Sie beginnt mit dem Befehl [newfont]:

```
[newfont]
Kennz_DtAlt
Kennz_DtAlt.bmp
Kennz_DtAlt_Alpha.bmp
19
2
```

Die erste Zeile „Kennz_DtAlt“ ist der Name des Fonts, wie er im [texttexture]-Befehl angegeben werden muss.

Es folgt die Bitmap, welche die Farbwerte enthält für den Fall, dass die Originalfarbe des Fonts angezeigt werden soll; in diesem Fall Kennz_DtAlt.bmp:



Da dieser Font vor allem für die Darstellung der Kfz-Kennzeichen verwendet wird, verfügt er über einen Bindestrich mit HU/ASU/Versicherungsplaketten, welche hier zu sehen sind. Der Rest ist natürlich schwarz.

In der nächsten Zeile wird die Alphakanal-Bitmap vermerkt, die bei den Fonts im Regelfall die wesentlich wichtigere ist:



Hier sind nun nicht nur alle Buchstaben und Zahlen sichtbar sondern auch der Bindestrich mit den Silhouetten der Plaketten.

Zu beachten ist, dass diese Bitmaps nur zur Erzeugung der Texttexturen dienen – sie selbst sind *keine* Texturen und dürfen daher von beliebiger Größe sein; es muss nur darauf geachtet werden, dass die Koordinatenangaben in der oft-Datei auch immer innerhalb der Bitmap liegen!

Im Anschluss an die beiden Bitmap-Einträge folgen die Höhe der Zeichen in Pixeln (hier 19) inklusive dem Abstand zur nächsten Reihe und die

Anzahl der Leerpixel, die zwischen die Zeichen gefügt werden sollen (hier 2).

Nach dieser Initialisierung folgen nun sämtliche Einträge für Buchstaben, die stets folgendermaßen aufgebaut sind:

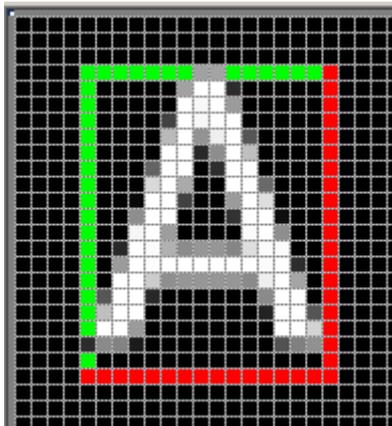
```
[char]  
A  
4  
19  
3
```

Zuerst der Buchstabe, der definiert werden soll. Wird auf Kleinbuchstaben verzichtet, verwendet OMSI automatisch den zugehörigen Großbuchstaben (außer bei Umlauten und „ß“ – wobei es beim „ß“ ja auch keinen zugehörigen Großbuchstaben im ANSI-Zeichensatz gibt! ;-)).

Der erste [char]-Eintrag ist automatisch das Zeichen, was immer dann verwendet wird, wenn das eigentlich geforderte Zeichen im Zeichensatz nicht vorhanden ist.

Es folgt dann die x-Koordinate vom linken Rand (inkl., 4) und die x-Koordinate vom rechten Rand (exkl., 19) und schließlich die Y-Koordinate der Oberkante (inkl., 3). Die der Unterkante ergibt sich automatisch über die bereits oben angegebene Höhe.

In der folgenden Graphik die gezoomte Darstellung des oben gezeigten Eintrages: Die grünen Kanten gehören mit zum Buchstaben, die roten nicht – daher oben die Bezeichnungen „inkl.“ und „exkl.“.



Bei einigen Fonts wird darauf verzichtet, eine eigene Bitmap für die Originalfarben mitzuliefern. Hier wird dann üblicherweise für beide Bitmaps dieselbe Datei angegeben.