ORCF Custom Content Guide

Für ORCF **2011.09p**

9. September 2011

Inhaltsverzeichnis

1 Custom Content und ORCF

ORCF wurde von Anfang an darauf ausgelegt, von den Usern um eigene Objekte erweitert zu werden – mit allen Möglichkeiten, die das Spiel bietet. Dazu ist diese Anleitung gut, auch wenn die garantiert noch nicht fertig ist, aber hey, das Spiel selbst ist ja auch noch alles andere als fertig und das einzige, was man bisher erstellen kann, sind einfache Objekte zum Hinstellen. Die können zwar auch animiert sein – dafür muss man aber etwas programmieren. Auf die Programmierung wird aber erst in einer zukünftigen Veröffentlichung eingegangen, wenn das System etwas ausgereifter ist.

1.1 Das Standardpaket

Alle Objekte, die in dieser Vorabversion enthalten sind, fliegen selbstverständlich raus. Die wurden nur erstellt um grafische Effekte zu demonstrieren und um etwas rumzuspielen.

Wenn ihr Lust habt, könnt ihr Sets für das so genannte Standardpaket erstellen, damit es standardmäßig bei ORCF mitgeliefert wird. Allerdings gibt es da neben einigen technischen Anforderungen, die später erwähnt werden, einige Voraussetzungen:

- Die Sets müssen unter einer Creative Commons-Lizenz stehen, die mindestens die unveränderte nichtkommerzielle Weitergabe mit Namensnennung erlaubt, vorzugsweise auch noch Veränderungen am Original, damit das Set nicht von den Erstellern selbst an technische Änderungen angepasst werden muss. Ein Beispiel ist die Lizenz CC BY-NC-SA 3.0.
- Um darauf hinzuweisen, was erlaubt ist, ist dem Set eine `readme.txt` beizulegen.
- Sie dürfen demnach keine Inhalte, die nicht unter einer zu der gewählten Creative Commons-Lizenz stehen, enthalten. Das gilt vor Allem für Texturen.

2 Voraussetzungen

2.1 Blender

Das mitgelieferte Export-Script für Objekte funktioniert nur mit Blender 2.5. Ihr benötigt also eine aktuelle Version von Blender sowie grundlegende Kenntnisse im Umgang mit der Software. Wenn ihr mit Blender 2.4 arbeitet, werdet ihr in 2.5 wahrscheinlich nicht sofort alles an gewohnter Stelle finden.

2.2 Das Export-Script

Im Unterordner `tools` befindet sich neben zwei ausführbaren Programmen die Datei `export.py`. Bevor ihr diese zu Blender hinzufügt, müsst ihr allerdings die Pfade zu einigen ORCF-eigenen Hilfsprogrammen angeben. Wenn ihr die Datei in einem Texteditor öffnet, der Zeilenumbrüche unterstützt (z. B. Kate, Notepad++, ...), findet ihr recht weit oben folgende Zeilen:

```
1 ocfgen_exe = '/home/philip/Delphi/orcf/tools/ocfgen'
2 orcf_data_path = '/home/philip/Delphi/orcf/data/'
3 orcf_personal_data_path = '/home/philip/orcf-data/'
```

Lasst euch nicht von den komischen Namen verwirren, ich schleppe den `Delphi`-Ordner schon seit gut 5 Jahren mit mir rum, ohne überhaupt noch Delphi zu benutzen – ihr müsst einfach die Pfade zu den entsprechenden Zielen angeben.

2.2.1 Windows

Unter Windows sieht das beispielsweise so aus:

```
1 ocfgen_exe = 'C:/orcf/tools/ocfgen.exe'
2 orcf_data_path = 'C:/orcf/data/'
3 orcf_personal_data_path = 'C:/orcf/data/'
```

Beachtet bitte, dass Python auch unter Windows einfache Schrägstriche bei Pfaden akzeptiert und dass die letzten beiden Pfade identisch sein müssen!

2.2.2 Linux

Unter Linux liegt der letzte Pfad immer im Userverzeichnis und heißt immer `orcf-data`. Hier ist also nur der Username zu ändern, während die anderen beiden Pfade dahin zeigen, wo das Programm liegt – vorzugsweise in `/opt/orcf`, wenn es für alle Systembenutzer zugänglich sein soll.

3 Ordnerstruktur eines Sets

Objekte liegen *immer* im `scenery`-Ordner in einem der Datenordner – es ist dabei völlig egal, ob es der persönliche Datenordner oder der Systemordner ist. Unter Windows sind die sowieso gleich. In der ersten Ebene liegen dabei die OCF-Dateien, die auf die einzelnen Objekte im Set verweisen, sowie die Ordner, in denen die eigentlichen Daten eurer Sets liegen. Damit es nicht zu Überschneidungen kommt, benennt eure Ordner und OCF-Dateien möglichst nach folgendem Schema:
`Name_des_Erstellers-Name_des_Sets` für den Ordner und
`Name_des_Erstellers-Name_des_Sets.ocf` für die OCF-Datei.

Innerhalb des Ordners seid ihr theoretisch völlig frei in der Namensgebung. Es sollten jedoch nur „normale“ Zeichen verwendet werden, d. h. Buchstaben, Zahlen sowie Unter- und Bindestriche. Leerzeichen führen jedoch zu Problemen.

Weil das Export-Script eine Menge Dateien erzeugt, die durchaus bei verschiedenen Objekten denselben Namen haben können, ist es empfehlenswert, für jedes Objekt einen eigenen Ordner anzulegen.

4 Objekte erstellen

Voraussetzung ist, dass ihr mit Blender modellieren und texturieren könnt. Hier sollen nur die Spezialitäten von Blender im Bezug auf ORCF-Objekte erläutert werden, nicht aber die Grundlagen, denn das würde zu viel Zeit in Anspruch nehmen – es gibt mehr als genug gute Blender-Tutorials da draußen.

4.1 Informationen über das Objekt

Die meisten Angaben hierzu werden in ORCF selbst angegeben – der Name des Autors steht allerdings in der Blender-Datei. Um diesen zu setzen, erstellt ihr im World-Panel ein Custom Property mit dem Namen AUTHOR und eurem Namen als Wert:

Diese Information wird bei jeder Datei benötigt, also kann es sich durchaus lohnen, ein Template zu erstellen, in dem das bereits drinsteht.

4.2 Materialien

Vorweg: Gebt euren Materialien aussagekräftige Namen! Jedes einzelne Material kann im Spiel vom Nutzer verändert werden (v. a. die Farbe), da hilft es wenig, wenn das Material „Material.005“ heißt. Leerzeichen sorgen aber für Probleme bei der OCF-Dateierstellung, verwendet also statt Leerzeichen bitte Unterstriche. Umlaute sind auch verboten, weil Windows Umlaute anders codiert als andere Systeme – hier gilt also dasselbe wie schon bei der Benennung der Dateien weiter oben.

Warnung: Nutzt Transparenz bitte nur, wenn nötig (z. B. bei Fenstern), denn das Rendering von transparenten Objekten ist in ORCF schlecht implementiert und führt zu Anzeigeproblemen.

4.2.1 Benutzbare Eigenschaften

- Materialfarbe (Diffuse, Diffuse →Intensity). Kann vom Spieler geändert werden.
- Emission (Shading →Emit). Aktiv für selbstleuchtende Materialien. Kann vom Spieler geändert werden.
- Glanzlicht-Intensität (Specular →Intensity)
- Glanzlicht-Härte (Specular →Hardness)
- Durchsichtigkeit (Transparency →Alpha)
- Reflexionsstärke (Mirror →Reflectivity)
- Reflexionsmodus (Mirror →Max Distance). Ein Wert von `0` bedeutet, dass *immer* eine fast statische Environment Map auf das Objekt gelegt wird (empfehlenswert bei Objekten mit geringer Reflexionsstärke sowie sehr großen Objekten), jeder größere Wert bedeutet, dass die Reflexionstextur dynamisch gerendert wird, wenn der User dies in den Grafikeinstellungen aktiviert hat
- Texturen, dazu aber mehr im Abschnitt 4.3 Texturen.

4.2.2 Custom Properties

- displacementHeight – betrifft nur User mit aktiviertem Parallax Occlusion Mapping. Gibt die maximal simulierte Tiefe in Metern an, Standardwert ist `0.04`, Werte bis `0.20` erzeugen in der Regel gute Ergebnisse.

4.3 Texturen

ORCF kann zwei verschiedene Typen von Texturen verarbeiten: Farbtexturen und Bump Maps. Bump Maps sind Texturen, die Informationen über die Ausrichtung und die Verschiebung von der Oberfläche eines Objekts enthalten, um den Detailgrad ohne zusätzliche Polygone deutlich zu erhöhen. Die Erstellung von Bump Maps mit Blender wird im Abschitt 5 Bumpmaps erläutert. **Hinweis:** Bitte sorgt dafür, dass eure Farbtexturen nur dann einen Alphakanal haben, wenn er wirklich genutzt wird! Sonst funktionieren Dinge wie Parallax Occlusion Mapping nicht und es kommt zu unschönen Artefakten.

4.3.1 Benutzbare Eigenschaften

- Typ muss `Image or Movie` sein
- Image →Source muss auf eine TGA-Datei verweisen
- Mapping →Coordinates muss auf `UV` stehen
- bei Influence kann entweder `Color` oder `Normal` aktiviert werden. Wenn `Color` aktiv ist, so wird die Farbe der Textur im Spiel mit der gewählten Materialfarbe multipliziert – wenn euer Objekt also einfärbbar sein soll, achtet darauf, dass eure Textur möglichst nur Grautöne beinhaltet. Wenn `Normal` aktiv ist, handelt es sich um eine Bumpmap.

4.4 Objekte

Das, was in ORCF als Mesh bezeichnet wird, heißt in Blender „Object“ und stellt die Geometrie dar.

4.4.1 Benutzbare Eigenschaften

- Position (Transform →Location). Ergibt sich, wenn man im Object Mode das Objekt verschiebt.
- Drehung (Transform →Rotation). Ergibt sich, wenn man im Object Mode das Objekt dreht.
- Elternobjekt ((Relation →Parent

). Alle Transformationsoperationen des Elternobjekts werden auch auf dieses Objekt übertragen, besonders bei Animationen interessannt.

Warnung: Die Skalierung (Transform →Scale) kann *nicht* benutzt werden! Wenn ihr skaliert, tut dies *immer* im Edit Mode.

4.4.2 Custom Properties

- MIN_DIST – falls mehrere LODs benutzt werden, gibt dies die minimale Distanz an, die das Objekt zum Betrachter haben muss, um an- gezeigt zu werden. Wenn keine LODs benutzt werden oder das Objekt den höchstmöglichen Detailgrad besitzt, sollte diese Eigenschaft nicht benutzt werden.
- MAX_DIST – falls mehrere LODs benutzt werden, gibt dies die maximale Distanz an, die das Objekt zum Betrachter haben muss, um an- gezeigt zu werden. Wenn keine LODs benutzt werden oder das Objekt den niedrigsten Detailgrad besitzt, sollte diese Eigenschaft nicht benutzt werden.

4.5 Lampen

Vorweg: Um eine Lampe in ORCF verwenden zu können, muss sie als Parent ein Objekt haben. Außerdem hat ORCF ein ähnliches Beleuchtungsmodell wie Blender, die genauen Berechnungen sind aber doch anders, deswegen sehen die Ergebnisse beim Rendern nicht zwangsläufig gleich aus.

4.5.1 Benutzbare Eigenschaften

- Position – ergibt sich, wenn die Lampe verschoben wird.
- Typ muss `Point` sein
- Schatten (SHADOW) muss auf `Ray Shadow` gesetzt werden, wenn die Lampe Schatten werfen soll.
- Farbe (LAMP) – wird in zukünftigen ORCF-Versionen auch vom Spieler geändert werden können. Erstellt daher bitte nicht für verschiedene Beleuchtungsfarben eigene Objekte.
- Energie (LAMP →ENERGY) – wird ebenfalls eines Tages vom Spieler geändert werden können.
- Leuchtweite (LAMP →ENERGY) – gibt an, in welcher Entfernung die Lampe nur noch die Hälfte ihrer ursprünglichen Leuchtkraft besitzt. Wird ebenfalls eines Tages vom Spieler geändert werden können.

4.5.2 Custom Properties

- ONLYNIGHT – der Wert spielt hier keine Rolle, wenn die Eigenschaft gesetzt ist, leuchtet die Lampe nur bei Nacht.

4.6 Ein paar Dinge...

ORCF ist nicht RCT3. Es beherrscht gerade in Sachen Platzierung viele Dinge, die RCT3 nicht kann. Daher solltet ihr folgendes beachten:

- Nutzt die Möglichkeit von Parallax Occlusion Mapping aus. Das spart oft sehr viele Polygone und kann den Detailgrad extrem erhöhen.
- Wenn ihr Alphamasken nutzt, um detailreiche Objekte mit einem einzelnen Polygon zu rendern, beachtet bitte, dass hier kein Parallax Occlusion Mapping funktioniert und dass an den Rändern der Textur (noch) Artefakte auftreten können.

- Objekte lassen sich beliebig drehen und spiegeln. Ihr braucht also für Seitenränder von Dächern nicht 2 Objekte erstellen, eines reicht.
- Bumpmaps und Farbtexturen benutzen dieselben Texturkoordinaten!
- In zukünftigen Versionen werden einfache Objekte wie Wände skalierbar sein, sodass der Eindruck entsteht, es stünden mehrere Wände nebeneinander. Die meisten Wände sollten deshalb nur ein Mal mit einer Größe von 2x2x0.2 Metern abgespeihert werden. **Hinweis:** Das gilt jedoch nicht für Wände, die sich nicht kacheln lassen – wenn also zum Beispiel eine Wand mit Basis erstellt wird, so würde eine Skalierung in die Höhe das Problem mit sich bringen, dass die Basis nicht nur direkt über dem Boden sichtbar ist. Hier ist es nötig, eine Wand ohne Basis zu erstellen, die sich dann wieder kacheln lässt.
- Verschiebt die Objekte so in der Höhe, dass die Stellen, die den Boden berühren sollen, auf dem Koordinatensystem von Blender die Höhe 0 haben. So lassen sich Objekte einfacher platzieren und stapeln.
- ORCF wird in zukünftigen Versionen die Möglichkeit erhalten, Objekte wie Zäune an schräge Stellen wie bspw. einen Terrainhügel oder einer Bogenbrücke auszurichten. Ihr braucht demnach keine Objekte erstellen, in denen diese Abschrägung bereits vorhanden ist.
- Benutzt LODs, wo es sinnvoll ist – aber bitte nicht zu viele. Drei Detailstufen sind in der Regel vollkommen ausreichend.
- Einfarbige Flächen benötigen keine Textur.

4.7 Linken von externen Ressourcen

Ihr könnt, um eine Menge Ressourcen zu sparen, Materialien und vor Allem Texturen aus vorher erstellten Blender-Dateien einbinden. Dazu geht ihr im File-Menü auf Link, wählt die Datei aus und sucht dort das entsprechende Material oder die entsprechende Textur. Diese lassen sich dann auf herkömmliche Weise auswählen, als wären sie in die Datei integriert.

Das sorgt dafür, dass die gelinkten Daten nicht mit in die OCF-Datei geschrieben werden und auch im Spiel nicht mehrfach geladen werden. Das spart Speicherplatz auf der Festplatte und auf der Grafikkarte, und gerade da sollte man sehr sparsam sein. Diese Funktionalität entspricht etwa den aus RCT3 bekannten Shared Textures. Nicht möglich ist das allerdings bei Objekten, die Geometrie muss also jedes mal mit exportiert werden.

5 Bumpmaps erstellen mit Blender

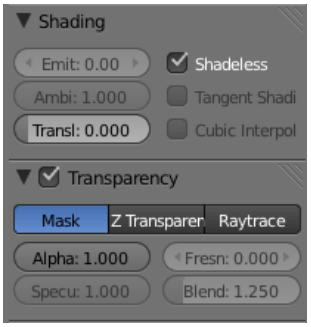
Hier soll kurz erklärt werden, wie man mit Blender eine Bumpmap aus einem Objekt erstellt.

5.1 Das Objekt

Das Objekt wird nicht ins Spiel importiert. Es dient lediglich dazu, Polygone in den eigentlichen Objekten einzusparen, ohne jedoch an Detailreichtum einzubüßen – daher muss hier bei diesem Objekt keine Rücksicht auf die Anzahl der Polygone genommen werden. Im Grunde erstellt ihr hier genau das, was ihr ansonsten in das Objekt mit eingebaut hättet, beispielsweise Dachziegel.

5.2 Das Material

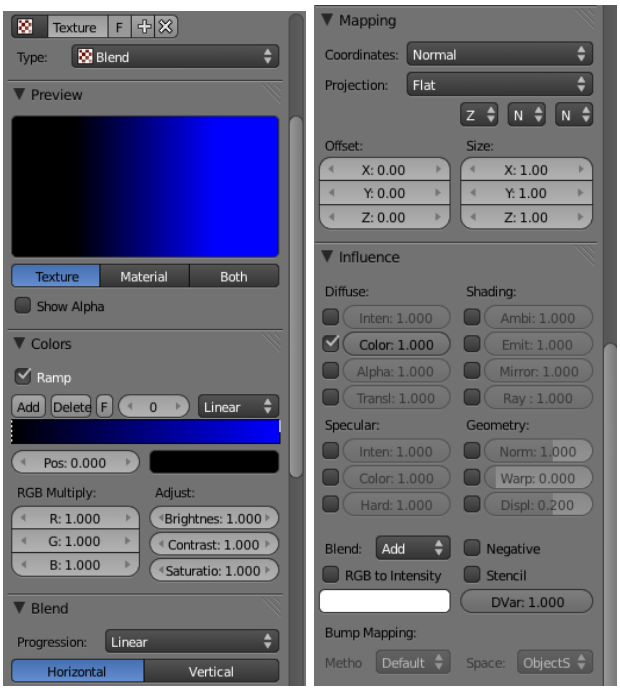
Ihr weist dem Objekt allerdings nicht die Farben zu, die das Objekt, das die Bumpmap nutzt, später haben soll. Der Trick liegt darin, das Objekt ohne Farben und ohne Beleuchtung zu rendern. Dazu färbt ihr das Material schwarz und aktiviert SHADING →SHADELESS, außerdem muss für die Höheninformationen noch TRANSPARENCY aktiviert und der Modus auf `Mask` gestellt werden.



5.3 Die Normalentexturen

Hier liegt eigentlich das Geheimnis. Blender kann mit einigen Tricks die Normale als Farbinformationen auf das Material legen. Dazu müsst ihr verstehen, wie eine Bumpmap eigentlich aussieht:

Jeder Pixel auf der Bumpmap repräsentiert einen Normalenvektor aus den Komponenten X, Y und Z. Die Werte liegen dabei alle zwischen -1 bis 1 – auf der Textur werden sie durch rote, grüne und blaue Farbkomponenten mit einem Wertebereich von 0 bis 255 dargestellt. Zunächst erstellt ihr also eine Textur für die Z-Komponente der Normale. Die ist blau und muss genau dieselben Eigenschaften haben wie unten dargestellt:



Wichtig ist, dass das Blau ein reines und volles Blau ist, also keine anderen Farbanteile mehr vorhanden sind.

Am Wichtigsten ist hier die Einstellung MAPPING →COORDINATES. Mit dem Wert `Normal` wird Blender angewiesen, die Normalen als Quelle für das Texturmapping zu verwenden.

Unter Mapping →Projection finden sich 3 Auswahlboxen. Die erste davon steht auf `Z`, die anderen beiden auf `N`. Damit teilen wir der Blender mit, dass nur der Z-Anteil der Normale berücksichtigt werden soll.

Achtet aber auch darauf, bei Influence →Blend `Add` auszuwählen, denn sonst seht ihr immer nur eine der drei Texturen.

Ihr müsst jetzt noch zwei weitere Texturen erstellen, eine rote für die X-Anteile und eine grüne für die Y-Anteile. Dazu muss lediglich die Farbe geändert werden und das Mapping ist von `Z` auf `X` bzw. `Y` zu stellen.

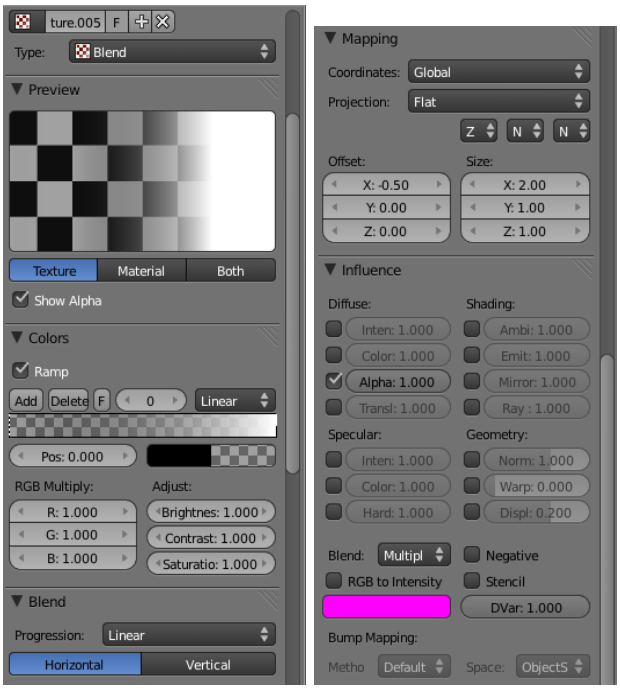
Am Ende sollte die Materialvorschau etwa folgendes ausspucken:

5.4 Die Höhentextur

Damit Parallax Occlusion Mapping mit dieser Bumpmap funktioniert, müsst ihr noch eine Alphatextur erstellen. Die ist sehr ähnlich zu den Normalentexturen, allerdings mit den folgenden Unterschieden:

- Die Textur geht nicht von Schwarz zu einer Farbe, sondern von voll durchsichtig zu weiß.
- Das Mapping benutzt nicht die Z-Komponente der Normale, sondern die Z-Position im World Space (dazu die Einstellung `Global` bei Mapping →Coordinates.

Alles in Allem sollte das etwa so aussehen:

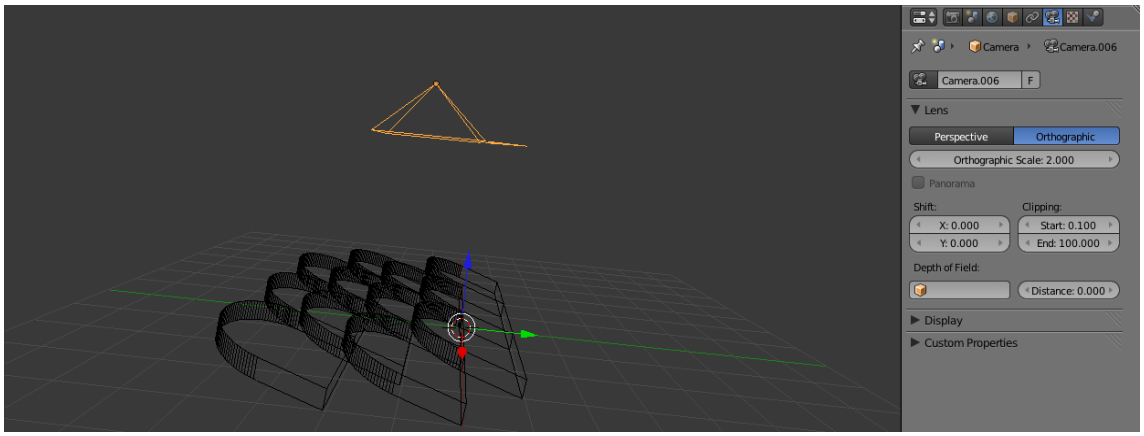


Offset und Size sind dabei so gewählt, dass für ein Objekt, dessen Z-Werte zwischen 0 und 1 liegen, eine perfekte Alpha-Map erstellt wird. Für andere Wertebereiche müsst ihr gegebenenfalls etwas rumexperimentieren, oft ist es aber am einfachsten, das Objekt an die richtige Stelle in

der Welt zu verschieben und zu skalieren. Skaliert euer Objekt aber keinesfalls ausschließlich in Z-Richtung, denn dadurch würden die Normalen verfälscht!

5.5 Die Kamera

Erstellt eine neue Kamera und setzt diese über das Objekt. Wie hoch sie liegt, ist egal, da sie nicht perspektivisch ist, sondern orthogonal. Das sollte dann etwa so aussehen:

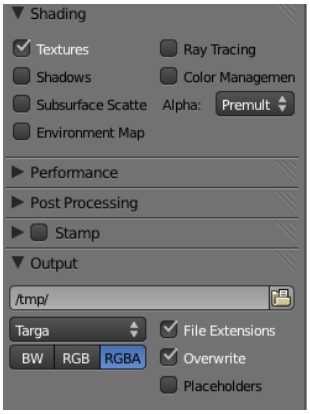


Im Scene-Panel könnt ihr die neue Kamera dann als Standardkamera festlegen, das ist unter Anderem für Rendervorgänge nötig.

Wie ihr Orthographic Scale wählt, hängt letztenendes von euerm Objekt ab. Von der Kamera aus sollte das gesamte Objekt sichtbar sein, ggf. ist es aber nötig, dass nur ein Teil sichtbar ist, damit die Textur kachelbar bleibt, wie hier im Beispiel mit den Dachziegeln.

5.6 Rendereinstellungen

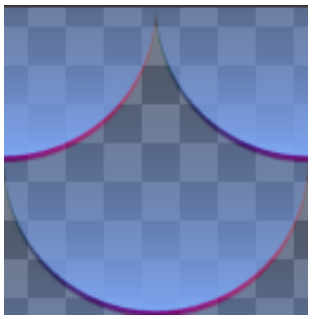
Damit die Bumpmap auch unverfälscht auf den Bildschirm gebracht wird, müsst ihr zunächst Blenders Farbkorrekturen abstellen und RGBA-Rendering aktivieren. Das geht mit den folgenden Einstellungen im Render-Panel:



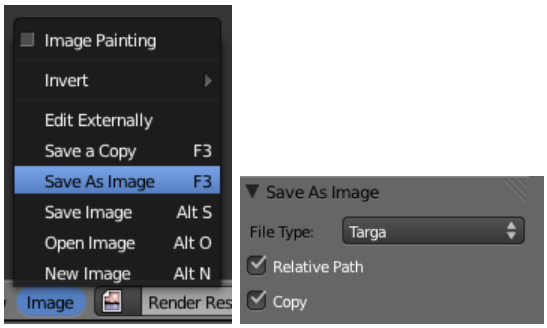
Wählt außerdem eine passende Größe, oftmals sind Bumpmaps von 256x256 Pixeln Größe völlig ausreichend.

5.7 Und.. Rendern!

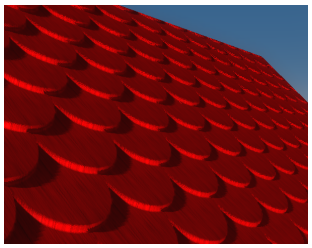
Nun sind alle Einstellungen getan und ihr könnt die Bumpmap rendern. Bei den Dachziegeln sieht diese etwa so aus, wenn man in Blenders Bildansicht die Alphakanaldarstellung aktiviert:



Das Bild muss nun noch gespeichert werden. Dazu wählt ihr im IMAGE-Menü den Eintrag SAVE AS IMAGE aus. Wichtig ist dabei, dass als Dateiformat `Targa` ausgewählt ist.

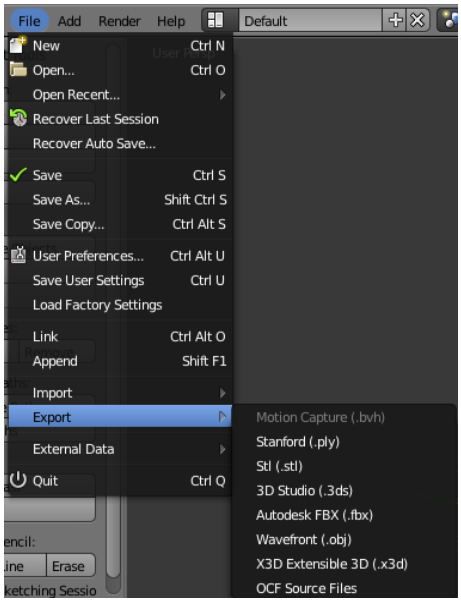


Und das Ganze dann im Spiel auf zwei Dreiecke gemappt:



6 Exportieren

Im Normalfall genügt ein Klick auf den entsprechenden Menüeintrag:



Solltet ihr eine Fehlermeldung erhalten, probiert folgendes:

- Speichert die Datei und ladet sie einmal neu. Das sorgt dafür, dass unbenutzte Datenblöcke wirklich gelöscht werden, und somit den Exporter nicht weiter stören.
- Wenn das nichts hilft, öffnet in Blender eine Pythonkonsole und gebt folgendes ein:

```
1 | bpy.ops.export_scene.ocf("/da/wo/eure/Blenderdatei/liegt/")
```

  und schaut auf die Ausgabe. Die Meldung kann auf fehlerhafte Eingaben zurückzuführen sein, aber auch auf einen Bug im Script.

- Sollte das Script durchlaufen, aber keine OCF-Datei in dem Ordner, wo auch eure Blenderdatei ist, auftauchen, so liegt das Problem bei der finalen Erstellung der OCF-Datei aus den Einzeldateien. Öffnet eine Kommandozeile in dem Ordner, in dem eure Blender-Datei liegt und führt die `.bat`-Datei (Windows) bzw. die `.sh`-Datei (Linux, Mac OS X) aus, die sich in dem Ordner befindet. Fehler dieser Art sind oft auf fehlerhafte Benennungen von jeglichen Objekten in Blender zurückzuführen, besonders Leerzeichen und Umlaute sorgen hier für Probleme, aber es kann genau so gut auch an einem Bug liegen.

7 Vorschaubilder

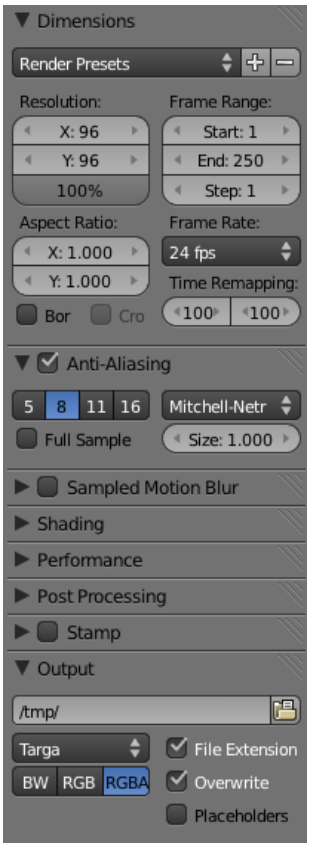
Damit der Spieler sofort in der Objektliste des Spiels sieht, wie welches Objekt aussieht, solltet ihr sinnvolle Vorschaubilder erstellen. Das geht auch direkt in Blender.

7.1 Allgemeines

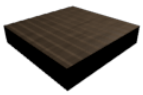
Es sollte auf dem Bild nur das Objekt zu sehen sein, und zwar so groß wie möglich. Wenn das Objekt nur mit einem anderen Objekt sinnvoll benutzt werden kann, so ist es ideal, wenn gezeigt wird, wie man das Objekt genau benutzt, indem zum Beispiel das zusätzlich benötigte Objekt ebenfalls integriert.

7.2 Rendereinstellungen

Die Größe eines Vorschaubilds in ORCF beträgt exakt 96x96 Pixel. Zwar funktioniert die Vorschau auch mit anderen Größen, allerdings sieht alles, was kleiner ist, ziemlich hässlich aus, und alles, was größer ist, ist Ressourcenverschwendung. Deswegen solltet ihr euch daran halten. Außerdem sollte die Vorschau einen transparenten Hintergrund haben, daher müsst ihr das RGBA-Rendering aktivieren. Gute Rendereinstellungen für ein Vorschaubild sehen beispielsweise so aus:



Ein Beispielbild eines nicht wirklich sehr gelungenen Holzbodens kann dann zum Beispiel so aussehen:



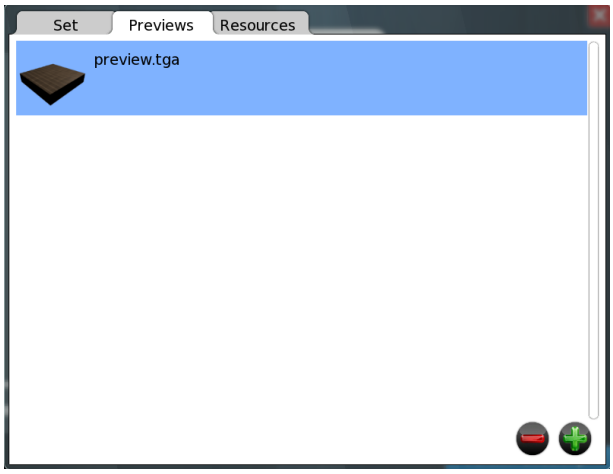
8 Der Set Creator

Damit eure Objekte auch im Spiel verwendet werden können, müsst ihr eine Set-Datei erstellen, die die Objekte referenziert. Erwähnt wurde diese bereits im Abschnitt 3 Ordnerstruktur eines Sets. Dazu gibt es in ORCF einen so genannten Set Creator, der allerdings etwas versteckt ist: Klickt im Hauptmenü auf einen leeren Bereich des Fensters und drückt die Taste S. Der Set Creator sollte sich öffnen.

Warnung: Der Set Creator ist bislang schlecht programmiert. Wenn ihr irgendwo vergesst, einen Namen, eine Beschreibung, eine Kategorie oder ein Vorschaubild auszuwählen, entsteht ein fehlerhaftes Set, das ihr weder benutzen noch nachträglich verändern könnt!

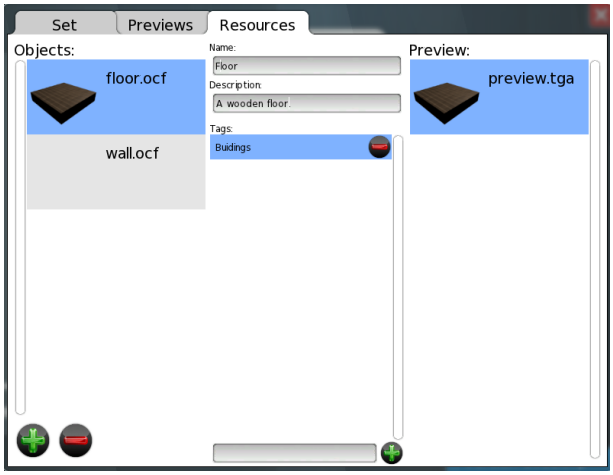
8.1 Vorschaubilder laden

Wählt den Reiter Previews aus. Die Oberfläche davon dürfte recht selbsterklärend sein: Mit dem grünen Plus könnt ihr TGA-Bilder laden und mit dem roten Minus ein ausgewähltes Bild wieder entfernen.



8.2 Objekte laden

Klickt auf den Reiter RESOURCES. Hier legt ihr fest, welche Objekte im Set vorhanden sind, wie sie heißen, wozu sie gut sind und in welchen Kategorien („Tags“) sie erscheinen sollen.



Die linke Liste enthält die Objekte. Mit dem grünen Plus könnt ihr OCF-Dateien hinzufügen, mit dem roten Minus wieder entfernen. Jedes Objekt hat einen Namen, eine Beschreibung, eine Liste von Kategorien und ein Vorschaubild. Die geladenen Vorschaubilder werden rechts angezeigt. Um eine dieser Eigenschaften für ein Objekt zu ändern, müsst ihr zunächst in der linken Liste das entsprechende Objekt auswählen.

8.2.1 Die Kategorien

Um auch bei einer großen Anzahl von Objekten die Übersicht zu wahren, kann der Nutzer verschiedene Kategorien anwählen. Diese sind allerdings nicht vom Spiel festgelegt, sondern von den Erstellern des Sets. Da es aber wenig hilft, wenn einer seine Teile in „Dächer“ einordnet und der nächste in „Dach“, solltet ihr gewisse Regeln beachten.

- Nutzt englische Begriffe. Nicht jeder spricht Deutsch.
- Nutzt bei Typenbeschreibungen die Mehrzahl, also „Walls“ statt „Wall“.
- Orientiert euch an existierenden Kategorien. Wenn schon Objekte in eine Kategorie „East Asia“ eingeordnet wurden, nutzt diese für entsprechende Objekte, anstatt eine neue Kategorie „China“ o. ä. einzurichten. Das wahrt die Übersicht.

- Gebt Kategorien für Themengebiete aussagekräftige Namen – ein Set mit nordischen Elementen macht sich besser in „Vikings“ als in „History“.
- Nutzt mehrere Kategorien. Ein Atlantis-Dach sollte sowohl in „Atlantis“ als auch in „Buildings“ und „Roofs“ erscheinen.

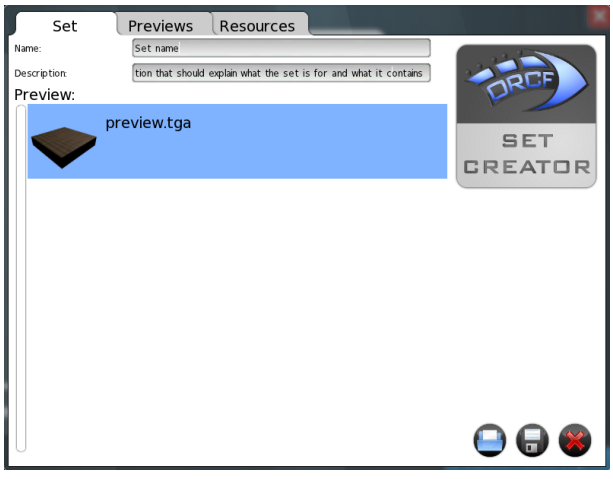
Im Folgenden mal eine Liste von Kategoriennamen, die eingehalten werden sollten:

- Buildings – Alles, was mit Gebäuden zu tun hat. Dächer, Wände usw.
- Walls – Alle möglichen Wände. Also auch Fenster, abgeschrägte Wände für Dächer, Ecken usw.
- Roofs – Dächer.
- Plants & Trees – Pflanzen und Bäume.
- Fences – Zäune.
- Lamps – Objekte mit Lampen.
- Particles – Für Objekte mit Partikeleffekten wie Feuer, Wasser, Staub, Nebel usw. (Noch nicht möglich)
- Sounds – Für Objekte, die einen Sound abspielen. (Noch nicht möglich)
- Flatrides – Für Flatrides. (Noch nicht möglich)
- Tracked Rides – Achterbahnen, Wasserbahnen etc. (Noch nicht möglich)
- Stalls – Läden und Stände. (Noch nicht möglich)

Ihr könnt neue Kategorien in die Liste hinzufügen, indem ihr den Namen der Kategorie in das Textfeld unten eintragt und auf das grüne Plus klickt. Die Kategorien, zu denen ein Objekt gehört, müssen einzeln ausgewählt werden und erscheinen in der Liste mit einem blauen Hintergrund.

8.3 Informationen zum Set

Im linken Reiter Set müsst ihr zu guter Letzt dem Set einen Namen geben. Verzichtet dabei bitte auf den Namen der Ersteller, diese werden schon in der Objektliste angezeigt.



Nun könnt ihr das Set abspeichern:

Beachtet dabei bitte das genannte Dateinamensschema. Der Name sollte dem des Set-Ordners gleichen, die Dateiendung wird automatisch angehängt.