# Blender to XNA Tutorial – Quick and Easy

Andrea Fuchsloch, Katja Weidner {fuchsloch|weidner}@fzi.de

Zentrum für Softwarekonzepte (ZfS) Karlsruhe

**Abstract**

Dreidimensionale Modelle werden in der Regel in einem 3D-Modellierungs-Tool wie bspw. Blender erstellt. Die Entwicklung von Spielen andererseits erfolgt in einer Entwicklungsumgebung wie dem XNA Game Studio. Die Integration der beiden Anwendungen stellt dabei eine Herausforderung für den Entwickler dar. Dieses Tutorial beschreibt, wie sich in Blender erstellte, dreidimensionale Modelle auf einfache Weise in das XNA Framework integrieren lassen. Auf das Ansprechen und Abspielen von Animationen die in Blender erstellt wurden, wird in diesem Tutorial nicht eingegangen.

Dreidimensionale Modelle bestehen in der Regel aus tausenden von Vertices. Diese von Hand festzulegen, ist nahezu unmöglich. Daher wird für das Erstellen von dreidimensionalen Modellen meist auf ein 3D-Modellierungs-Tool zurückgegriffen, mit dessen Hilfe komplexe Körper erstellt und exportiert werden können [3]. Das XNA Framework, bzw. um genau zu sein die Content Pipeline, unterstützt standardmäßig zwei Formate: Das Autodesk FBX Format (.fbx) [1] und das DirectX File Format (.x) [4].

Im Folgenden wird auf die Kombination aus dem 3D-Modellierungs-Tool Blender und dem XNA Game Studio eingegangen. Unter www.blender.org stehen sowohl das Tool als auch der Quellcode zum Download bereit [7].

An dieser Stelle soll auch nicht weiter auf das Erstellen von dreidimensionalen Modellen mit Hilfe von Blender eingegangen werden, sondern vielmehr erläutert werden, wie sich diese auf einfache Weise in das XNA Framework einbinden lassen. Anhang I beinhaltet eine Sammlung an Webseiten, die für die Erstellung von dreidimensionalen Modellen mit Blender als besonders hilfreich erachtet werden.

## 1 Grundlagen

Wurde ein Modell mit Hilfe von Blender erstellt, muss es in eines der bereits erwähnten Formate exportiert werden. Bevor das Modell aber exportiert wird, sollte folgendes beachtet werden: Blender arbeitet weder mit einem rechtshändigen Koordinatensystem, wie dies vom XNA Framework verwendet wird (siehe Abbildung 1: X-Achse nach rechts, Y-Achse nach oben und die Z-Achse nach vorne), noch mit einem linkshändigen Koordinatensystem (siehe Abbildung 2: X-Achse nach rechts, Y-Achse nach oben und die Z-Achse nach hinten) wie dies bei DirectX der Fall ist.

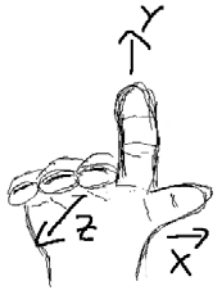


**Abbildung 1: Rechte-Hand-Koordinatensystem.**

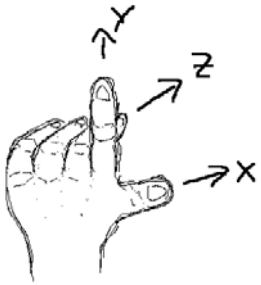


**Abbildung 2: Linke-Hand-Koordinatensystem.**

In Blender zeigt die X-Achse nach rechts, die Y-Achse nach vorne und die Z-Achse nach oben. Dieser ungewöhnlichen Anordnung der Achsen könnte die Überlegung zugrunde liegen, dass die zweidimensionale Ansicht, bei der die X-Achse für die Breite und die Y-Achse für die Höhe steht, flach auf dem von Blender zur Verfügung gestellten Raster liegt und mit der Z-Achse die Höhe bzw. Tiefe hinzukommt. Dieses von Blender verwendete Koordinatensystem wird in Abbildung 3 skizziert.

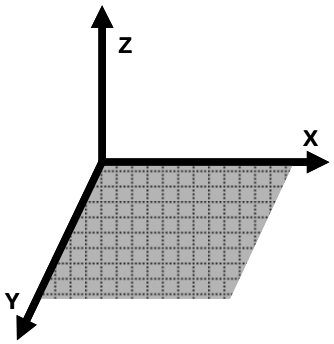


**Abbildung 3: Anordnung der Achsen in Blender.**

## 2 Vorbereitung

### 2.1 Modell rotieren

Um ein Modell möglichst einfach aus Blender in XNA einbinden zu können, bietet es sich an, das Modell bereits in Blender in ein rechtshändiges Koordinatensystem zu rotieren. Die X-Achse muss demzufolge nach rechts, die Y-Achse nach oben und die Z-Achse nach vorne zeigen, wie dies Abbildung 4 zu entnehmen ist. Diese Ansicht bietet Blender im Window Type „3D View“ unter „View“ → „Top“. Abbildung 5 zeigt den Pfad zu dieser Einstellung.

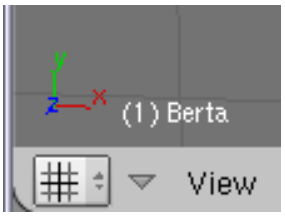


**Abbildung 4: Rechtshändiges Koordinatensystem in Blender.**

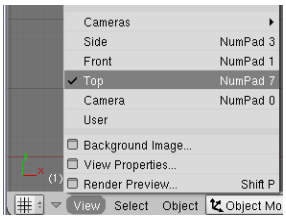


**Abbildung 5: Auswahl der „Top" Ansicht in Blender.**

Entgegen der Bezeichnung „Top“ sollte diese Ansicht nach dem Rotieren die Front-Ansicht des Modells zeigen, und nicht wie von Blender vorgesehen die Ansicht von oben. Um das Modell in die gewünschte Position zu rotieren bietet es sich an, in den „Objekt Mode“ zu wechseln. Das ausgewählte Objekt lässt sich mit Hilfe der „R“-Taste rotieren. Wird nach der „R“-Taste die „X“, „Y“ oder „Z“-Taste gewählt, erfolgt eine Rotation um die entsprechende Achse [2].

### 2.2 Pivot Punkt setzen

Nachdem das Modell in die Anordnung eines rechtshändigen Koordinatensystems rotiert wurde, sollte die Position des Modells angepasst werden. Für das Exportieren bietet sich folgende Position an: Der Ursprung des Koordinatensystems sollte gleichzeitig die Mitte der Unterseite des Modells sein. Das Modell sollte demnach auf der X-Achse „stehen“, und somit nur den positiven Bereich der Y-Achse verwenden. An der X- und Z-Achse wird das Modell jeweils zentriert zum Ursprung ausgerichtet.

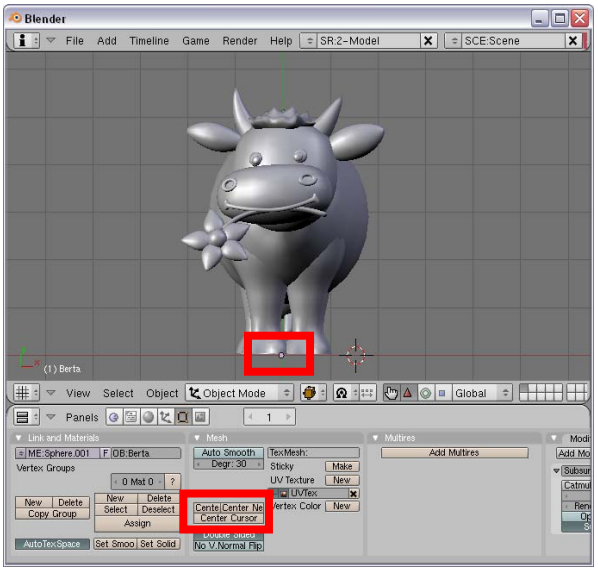


**Abbildung 6: Anordnung des Modells vor dem Export.**

Wurde das Modell nun „auf“ den Ursprung positioniert, sollte auch der Pivot-Punkt (das ist der kleine lila Punkt in Blender), der den Schwer- bzw. Ausgangspunkt für viele Transformationen darstellt, auf den Ursprung gelegt werden [2]. Dies lässt sich am einfachsten erreichen, indem mit dem Cursor der Ursprung des Koordinatensystems ausgewählt wird. Dabei ist darauf zu achten, dass auch die Z-Achse im Ursprung ausgewählt ist. Nachdem der Cursor in diese Position gebracht wurde, sollte der Pivot Punkt an die Stelle des Cursors gesetzt werden. Dazu einfach im „Object Mode“ im Window Type „Buttons Windows“ unter „Panels“ -> „Editing“ -> „Mesh“ den Button „Center Cursor“ klicken. In Abbildung 6 wurde der Pivot Punkt im Ursprung des Koordinatensystems hervorgehoben, und der „Center Cursor“ Button farblich markiert.

Um sicher zu stellen, dass diese Eigenschaften der Transformation auch gesetzt werden, ist es

hilfreich, alle Meshes des Modells auszuwählen und im Window Type „3D View“ die Funktion „Object“ -> „Clear/Apply“ -> „Apply Scale/Rotation“ (siehe Abbildung 7) anzuwenden.

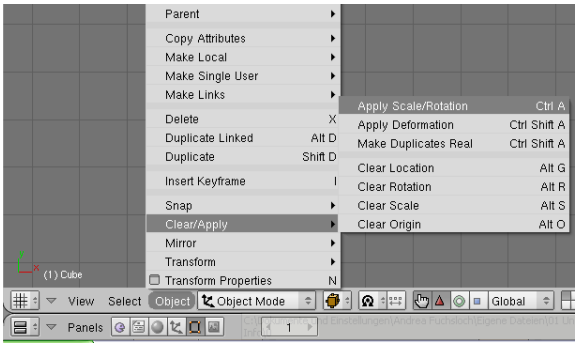


**Abbildung 7: Die Eigenschaften der Transformation fixieren.**

## 2.3 Modell texturieren

In der Regel werden dreidimensionale Modelle zusammen mit einer Textur verwendet. Eine Textur ermöglicht es, die Oberfläche eines 3D-Modells mit einem zweidimensionalen Bild zu gestalten. Für die Modellierung mit Blender bieten sich zwei Möglichkeit für die Erstellung einer Textur an: Zum einen kann diese direkt in Blender erstellt werden und zum anderen ist es möglich, die Textur eines Modells in einem Bildbearbeitungsprogramm wie bspw. Gimp oder Photoshop zu erstellen. In jedem Fall geht der Erstellung einer Textur aber das so genannte UV-Mapping voraus. Die Erstellung einer Textur wird im Folgenden nur grob umrissen. Anhang II beinhaltet Verweise auf Webseiten, die ausführliche Informationen zu diesem Thema bieten.

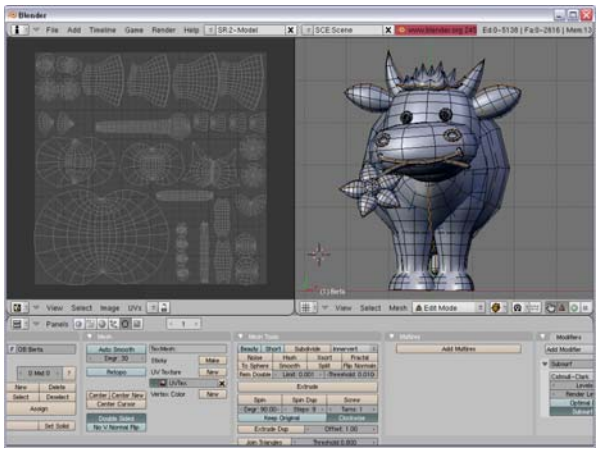

**Abbildung 8: Abrollen der Meshes entlang der definierten Schnittkanten.**

Das UV-Mapping ermöglicht es, dreidimensionale Modelle auf ein zweidimensionales Bild zu projizieren. Das dreidimensionale Modell wird dabei entlang definierter Schnittkanten, den so genannten Seams, auseinandergefaltet und auf ein zweidimensionales Bild übertragen. Abbildung 8 zeigt links das entstandene zweidimensionale Bild und rechts das dreidimensionale Modell. Jedem Vertex werden dabei zweidimensionale Koordinaten zugewiesen, die auch UV-Vertices oder UVs genannte werden [2]. Dieser Vorgang wird in Blender als „unwrap“ bezeichnete. Die auf diese Weise erstellte UV-Map kann beispielsweise im TGA Format exportiert, und in einem beliebigen Bildbearbeitungsprogramm weiter gestaltet werden. Die erstellte Textur kann anschließend wieder in Blender geöffnet und „über“ das Modell gelegt werden. Abbildung 8 zeigt links die Textur die „über“ das Modell auf der rechten Seite gelegt wurde.

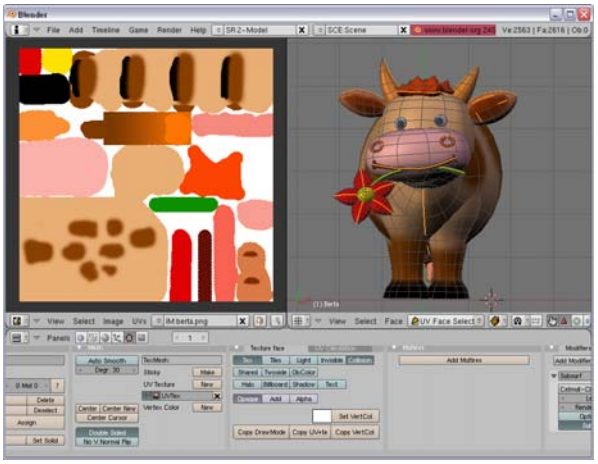

**Abbildung 9: Das 3D-Modell und seine Textur in Blender.**

# 3 Export des Modells

Wurden diese Vorarbeiten durchgeführt, kann das Modell exportiert werden. Zu diesem Zweck bietet es sich an, in den „Objekt Mode“ zu wechseln und das Modell auszuwählen. Der Aufruf der Export-Skripte ist über den Menüpunkt „File“ -> „Export“ zu erreichen. Für das FXB Format ist der Eintrag „Autodesk FBX (.fbx)“ und für das X File Format der Eintrag „DirectX (.x)“ auszuwählen. Da das Modell bereits in ein rechtshändiges Koordinatensystem gebracht wurde, sind beim Export in das FBX Format keinerlei Rotationen mehr erforderlich. Abbildung 10 zeigt die Einstellung beim Export in das FBX Format.


**Abbildung 10: Einstellungen für den Modell-Export im FBX Format.**

Wie bereits erwähnt verwendet das X File Format ein linkshändiges Koordinatensystem, daher besteht die Möglichkeit, beim Export die Z-Achse umzukehren. Die Option „Flip Z“, die standardmäßig aktiviert ist und somit zu einem linkshändigen System führt, sollte deaktiviert werden, um rechtshändiges Koordinatensystem zu erhalten.

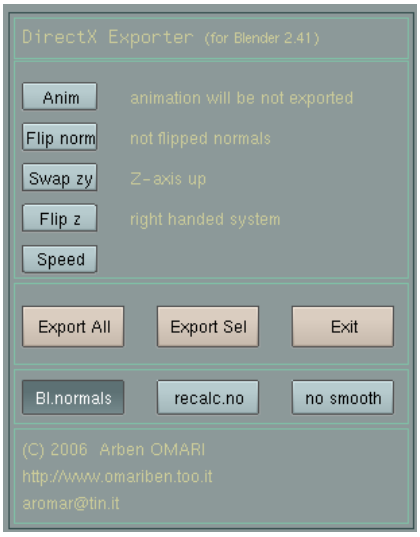


**Abbildung 11: Einstellungen für den Modell-Export im X File Format.**

## 4 Nacharbeiten am exportierten Modell

Modelle, die auf diese Weise exportiert wurden, „kennen“ ihre zugehörige Textur. Sie beinhalten Verweise auf Ihre Texturen in Form von relativen Pfadangaben. Dies kann für die Verwendung innerhalb des XNA Frameworks ungünstig sein, da der Entwickler den Speicherort der Modelle und Texturen in der Regel selbst bestimmen möchte. Die folgende Abbildung 12 zeigt eine gängige Struktur des Content Verzeichnisses.

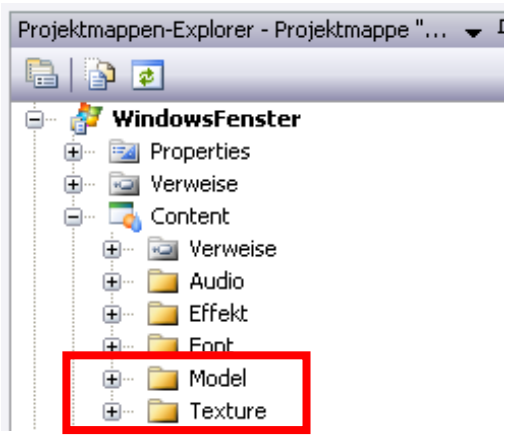


**Abbildung 12: Mögliche Organisation einer Projektmappe im XNA Game Studio**

Eine einfache Möglichkeit dieses Problem zu umgehen besteht darin, diese Verweise innerhalb der Modelle aus zu kommentieren oder zu löschen. Beide Formate (FBX und X File) können mit einem einfachen Editor problemlos geöffnet werden. Die verwendete Syntax ähnelt der Programmiersprache C++ und verwendet dieselben Kommentarzeichen. Im DirectX File Format muss lediglich die folgende Zeile aus der Modelldatei auskommentiert werden:

```
//TextureFilename { "<DateinameTextur>"; }
```

Für das FBX Format ist dies etwas aufwendiger, denn dieses Format enthält mehrere Verweise auf die Texturen:

Im Abschnitt Object definitions sind dies die beiden Funktionen:

```
Video: "Video:: <DateinameTextur>", "Clip" { ... }

Texture: "Texture:: <DateinameTextur>", "TextureVideoClip" { ... }
```

Im Abschnitt Object relations die beiden Funktionen:

```
Texture: "Texture::<DateinameTextur>", "TextureVideoClip" {}

Video: "Video::<DateinameTextur>", "Clip" {}
```

und im Abschnitt Object connections folgende Zeilen:

```
Connect: "OO", "Texture:: <DateinameTextur>", "Model::Sphere.001"

Connect: "OO", "Video:: <DateinameTextur>", "Texture:: <DateinameTextur>"
```

Wurden diese Verweise innerhalb des Modells umgangen, können Modell und Textur unabhän-

gig voneinander im XNA Framework verwaltet werden. Die beiden Objekte werden erst beim Zeichnen zusammengeführt.

## 5 Laden eines Modells in XNA

Die beiden erstellten Dateien, das Modell und die Textur, werden zunächst dem Projekt hinzugefügt. Hierfür sollte innerhalb des XNA Grame Studios, im Fenster „Projektmappen-Explorer“ das Kontextmenü des Zielverzeichnisses geöffnet werden. Mit Hilfe der Menüpunkte „Add“ -> „Vorhandenes Element“ öffnet sich das Dialogfenster „Vorhandenes Element hinzufügen“. Evtl. ist es erforderlich, im Dialogfenster den Dateityp „Content Pipeline Files“ auszuwählen damit die Modell- und Texturdateien in der Verzeichnisstruktur angezeigt werden.

Für das Modell muss nun im „Eigenschaften“ Fenster ein Importer bzw. ein Processor der Content Pipeline gewählt werden. Entsprechend dem Format sollte für den Content Importer „Autodesk FBX – XNA Framework“ oder „X File – XNA Framework“ gewählt werden. Als Content Processor sollte für beide Formate der Eintrag „Model – XNA Framework“ gewählt werden. Abbildung 13 zeigt die oben beschriebenen Einstellungen für das FBX Format. Die Konfiguration der Content Pipeline wird für die beiden Standardformate weitgehend von XNA übernommen.

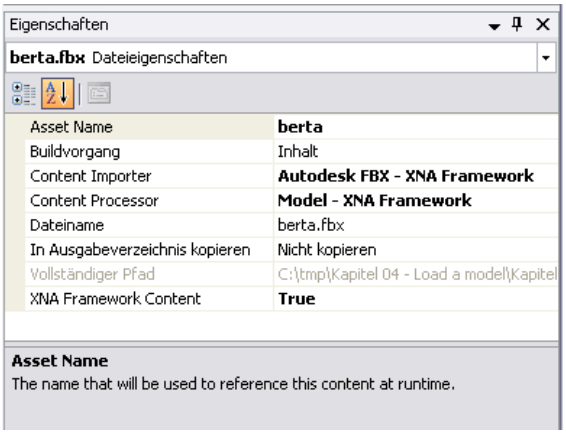


**Abbildung 13: Eigenschaften des Modells in XNA.**

Grundsätzlich unterstützt die Content Pipeline jedes Format, für das ein geeigneter Content Importer sowie ein geeigneter Content Processor vorliegen [3]. Das Laden eines Modells erfolgt analog zum Laden einer Textur über die `Load()` – Methode des `ContentManager` Objekts [5]. Anhang III zeigt in Zeile 071ff. ein Beispiel für das Laden des Modells sowie der Textur.

## 6 Rendern eines Modells in XNA

Das Rendern eines Modell wird in dem von Microsoft angebotenen Tutorial „How To: Render a Model“ sehr anschaulich beschrieben [6]. Daher wird an dieser Stelle nicht näher auf das Rendern eines Modells eingegangen, sondern auf die Besonderheit beim Rendern eines texturierten Modells.

Jedes Modell besteht aus einem oder mehreren Meshes, die wiederum aus mehreren Teilen bestehen können. Für jedes dieser Teilobjekte existiert ein Effekt, wie beispielsweise der `BasicEffect`. Anhang III instanziiert diesen in Zeile 138. Effekte bestehen in der Regel aus einem Vertex Shader, einem Pixel Shader und einer Technique. Um ein texturiertes Modell zeichnen zu können, muss die Textur dem Effekt übergeben werden. Dies kann beispielsweise mit Hilfe der `Texture` Property des `BasicEffect` Objektes erreicht werden, wie dies in Anhang III, Zeile 139 zu sehen ist [5]. Da jedes Teilobjekt über einen eigenen Effekt verfügen kann, ist es auch möglich für jeden Teil eine andere Textur festzulegen. Anhang III beinhaltet ein Beispiel für das Zeichnen eines texturierten Modells. Abbildung 14 zeigt das Ergebnis der in diesem Tutorial beschriebenen Vorgehensweise: Ein in Blender erstelltes und mit Hilfe des XNA Frameworks gerendertes, texturiertes Modell.

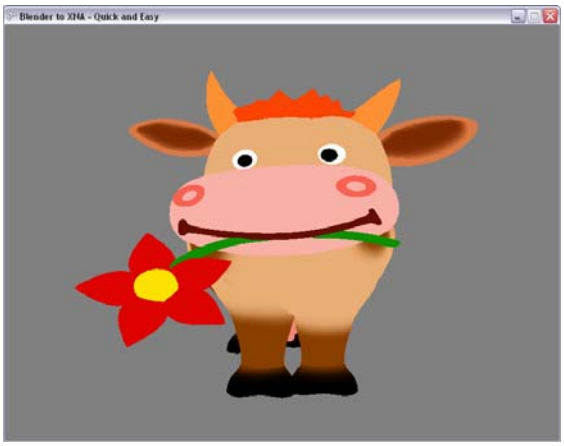


**Abbildung 14: Rendering eines texturierten Modells im XNA Framework.**

## 7 Zusammenfassung

Anhand eines durchgehenden Beispiels wurde gezeigt, wie sich dreidimensionale Modelle, die im kostenfreien 3D-Modellierungstool Blender erstellt

wurden, in das kostenfreie XNA Game Studio einbinden lassen. Das gesamte Beispiel, einschließlich der Blenderdatei (.blend), dem bearbeiteten FBX Format (.fbx), der Modelltextur (.png) sowie das XNA Game Studio Projekt, kann von der Webseite des Zentrums für Softwarekonzepte (ZfS) Karlsruhe heruntergeladen werden [8]. Die Verwendung dieser Dateien ist gestattet, sofern folgender Urheberrechtshinweis hinzugefügt wird: „Copyright (c) 2008, Zentrum für Softwarekonzepte (ZfS) Karlsruhe“.

## 8 Referenzen


[1] Autodesk Corporation: „Overview FBX Format“ Homepage: http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=8224901.

[2] Blender Dokumentation – Ein Wikiprojekt der deutschsprachigen Blendercommunity in Wikibooks. Homepage: http://de.wikibooks.org/wiki/Blender_Dokumentation.

[3] Konerow, Jens: „XNA Framework – schnell + kompakt“, entwickler.press, Software & Support Verlag GmbH, Frankfurt, 2007.

[4] MSDN - DirectX Developer Center: „X File Format Reference“. Homepage http://msdn2.microsoft.com/en-us/library/bb173014(VS.85).aspx.

[5] MSDN - Library für Visual Studio 2005 Express Editions. Homepage: ms-help://MS.VSExpressCC.v80/MS.NETFramework.v20.de/dv_vsref/html/259dfa46-a3bd-4b77-8acb-1cfab3eaae5a.htm.

[6] MSDN - XNA Developer Center: „How To: Render a Model“. Homepage: http://msdn2.microsoft.com/en-us/library/bb203933.aspx.

[7] Stichting Blender Foundation Amsterdam. Homepage: http://www.blender.org/.

[8] Zentrum für Softwarekonzepte Karlsruhe. Homepage: http://zfs.fzi.de.

**Anhang I**

Links zur Modellierung mit Blender.

Für die Erstellung dreidimensionaler Modelle mit Hilfe von Blender gibt es zahlreiche Tutorials im Internet. Im Folgenden sind einige hilfreiche Links aufgelistet:

1.) Das deutschsprachige Blender Handbuch: http://de.wikibooks.org/wiki/Blender_Dokumentation

2.) Besonders zu empfehlen! Das „Pfefferkuchenmann“ Tutorial: http://de.wikibooks.org/wiki/Blender_Dokumentation:_Die_erste_Animation_in_30_plus_30_Minuten.

3.) Ein englischsprachiges Buch zu Blender, das in die Arbeit mit Blender einführt: http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro

4.) Weitere hilfreiche Tutorials zu Blender: http://www.blendernation.com/tutorials/

5.) Videotutorials für das fortgeschrittene Arbeiten mit Blender: http://www.montagestudio.org/tut/

6.) „Dragonrose“ Serie auf YouTube mit insgesamt neun Schritten. Erstes Tutorial: http://de.youtube.com/watch?v=oIuPUhht5-M

**Anhang II**

Links zur Erstellung von Texturen.

1.) Der Vorgang des UV-Mappings wird unter http://de.wikibooks.org/wiki/Blender_Dokumentation:_UV-Mapping sehr anschaulich beschrieben.

2.) Eine Einführung zum Arbeiten mit Texturen ist zu finden unter: http://de.wikibooks.org/wiki/Blender_Dokumentation:_Texturen.

3.) Unter http://de.wikibooks.org/wiki/Blender_Dokumentation:_Suzanne_mit_UV-Mapping wird der Einsatz von Schnittkanten (Seams) für das UV-Mapping an einem Beispiel dargestellt.

4.) Ein vollständiges Beispiel - vom Setzen der Seams bis zum Anlegen der Textur in Blender: http://de.wikibooks.org/wiki/Blender_Dokumentation/_Tutorials/_Texturen/_UV-Texturierung_eines_Kopfes

**Anhang III**

Laden und Rendern eines Modells in XNA

```
001     #region Using Statements
002     using System;
003     using System.Collections.Generic;
004     using Microsoft.Xna.Framework;
005     using Microsoft.Xna.Framework.Audio;
006     using Microsoft.Xna.Framework.Content;
007     using Microsoft.Xna.Framework.Graphics;
008     using Microsoft.Xna.Framework.Input;
009     using Microsoft.Xna.Framework.Storage;
010     #endregion
011
012     namespace Blender2XNA
013     {
014         /// <summary>
015         /// This is the main type for your game
016         /// </summary>
017         public class Game1 : Microsoft.Xna.Framework.Game
018         {
019             private GraphicsDeviceManager graphics;
020             private ContentManager content;
021
022             private Matrix worldMatrix = Matrix.Identity;
023             private Matrix viewMatrix = Matrix.Identity;
024             private Matrix projMatrix = Matrix.Identity;
025
026             private Model model = null;
027             private Texture2D texture = null;
028
029
030             public Game1()
031             {
032                 graphics = new GraphicsDeviceManager(this);
033                 content = new ContentManager(Services);
034             }
035
036
037             /// <summary>
038             /// Allows the game to perform any initialization it needs to before starting to run.
039             /// This is where it can query for any required services and load any non-graphic
040             /// related content.  Calling base.Initialize will enumerate through any components
041             /// and initialize them as well.
042             /// </summary>
043             protected override void Initialize()
044             {
045                 // TODO: Add your initialization logic here
046                 base.Window.Title = "Blender to XNA – Quick and Easy Tutorial";
047
                    float aspectRatio = graphics.GraphicsDevice.Viewport.Width /
048                                        graphics.GraphicsDevice.Viewport.Height;
049
```

```
050             //View Matrix
                viewMatrix = Matrix.CreateLookAt
051                         (new Vector3(0.0f, 50.0f, 2000), Vector3.Zero, Vector3.Up);
052
053             //Projection Matrix
                projMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45.0f),
054                         aspectRatio, 1.0f, 10000.0f);
055
056             base.Initialize();
057         }
058
059
060         /// <summary>
061         /// Load your graphics content.  If loadAllContent is true, you should
062         /// load content from both ResourceManagementMode pools.  Otherwise, just
063         /// load ResourceManagementMode.Manual content.
064         /// </summary>
065         /// <param name="loadAllContent">Which type of content to load.</param>
066         protected override void LoadGraphicsContent(bool loadAllContent)
067         {
068             if (loadAllContent)
069             {
070                 // TODO: Load any ResourceManagementMode.Automatic content
071                 //Model laden
072                 model = this.content.Load<Model>("Content\\Models\\berta");
073                 //Textur laden
074                 texture = this.content.Load<Texture2D>("Content\\Textures\\berta");
075             }
076
077             // TODO: Load any ResourceManagementMode.Manual content
078         }
079
080
081         /// <summary>
082         /// Unload your graphics content.  If unloadAllContent is true, you should
083         /// unload content from both ResourceManagementMode pools.  Otherwise, just
084         /// unload ResourceManagementMode.Manual content.  Manual content will get
085         /// Disposed by the GraphicsDevice during a Reset.
086         /// </summary>
087         /// <param name="unloadAllContent">Which type of content to unload.</param>
088         protected override void UnloadGraphicsContent(bool unloadAllContent)
089         {
090             if (unloadAllContent)
091             {
092                 // TODO: Unload any ResourceManagementMode.Automatic content
093                 content.Unload();
094             }
095
096             // TODO: Unload any ResourceManagementMode.Manual content
097         }
098
099
100         /// <summary>
101         /// Allows the game to run logic such as updating the world,
102         /// checking for collisions, gathering input and playing audio.
```

```
103         /// </summary>
104         /// <param name="gameTime">Provides a snapshot of timing values.</param>
105         protected override void Update(GameTime gameTime)
106         {
107             // Allows the game to exit
108             if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
109                 this.Exit();
110 
111             // TODO: Add your update logic here
112 
113             base.Update(gameTime);
114         }
115 
116 
117         /// <summary>
118         /// This is called when the game should draw itself.
119         /// </summary>
120         /// <param name="gameTime">Provides a snapshot of timing values.</param>
121         protected override void Draw(GameTime gameTime)
122         {
123             graphics.GraphicsDevice.Clear(Color.Gray);
124 
125             graphics.GraphicsDevice.RenderState.CullMode = CullMode.None;
126 
127             Matrix[] transMatrix = new Matrix[model.Bones.Count];
128             model.CopyAbsoluteBoneTransformsTo(transMatrix);
129 
130             //Für jedes Mesh im Modell
131             for (int i = 0; i < model.Meshes.Count; i++)
132             {
133                 ModelMesh mesh = model.Meshes[i];
134 
135                 //Für jeden Effekt im Mesh
136                 for (int j = 0; j < mesh.Effects.Count; j++)
137                 {
138                     BasicEffect basicEffect = mesh.Effects[j] as BasicEffect;
139                     basicEffect.TextureEnabled = true;
140                     //basicEffect.EnableDefaultLighting();
141                     basicEffect.View = viewMatrix;
142                     basicEffect.Projection = projMatrix;
143                     basicEffect.World = worldMatrix * transMatrix[mesh.ParentBone.Index];
144                     basicEffect.Texture = texture;
145                 }
146                 mesh.Draw();
147             }
148 
149             base.Draw(gameTime);
150         }
151     }
152 }
153 
```