2016
2
ISSN 1433-2620 > B 43362 >> 20. Jahrgang >>> www.digitalproduction.com
Published by ATEC

Deutschland € 15,20
Österreich € 17,–
Schweiz sfr 23,–

DIGITAL PRODUCTION

DIGITAL PRODUCTION

MAGAZIN FÜR DIGITALE MEDIENPRODUKTION

MÄRZ | APRIL 02:2016

Fokus: Pipelines

Back-ups & Asset-Management-Grundlagen, Tools & Tricks

Zoomania

Kinorunde mit Star Wars, Disney und The Martian

Praxis satt!

4K-Monitore im Test, Quixel, Houdini Engine, Sony A7R II

4 194336 215200 02

Echtzeit-Produktvisualisierung & PBR in Modo und Marmoset

In diesem Artikel erstellen wir ein 3D-Modell, welches in Echtzeit gerendert wird. Dabei werden in den Workflow State-of-the-Art-Videospieltechnologien und Konzepte mit einbezogen.

von Patrick Möchel

Bei Produktvisualisierungen bietet Echtzeit einige Vorteile: Änderungen sind sofort sichtbar, lange Renderzeiten fallen weg und auf teure Spezialhardware kann verzichtet werden. In der Vergangenheit waren zwar teure Hard- und Softwarelösungen für fotorealistisches Rendern notwendig, aber dies ist spätestens seit der Einführung von Physically Based Rendering (PBR) im Bereich der Videospielgrafik nicht mehr der Fall.

Zum Hintergrund: Für das Berechnen von fotorealistischen Bildern werden im Computer Naturgesetze simuliert. Für die korrekte Darstellung von Licht, Schatten und Oberflächen sind komplexe mathematische und physikalische Berechnungen notwendig. Dies schlägt sich in langen Renderzeiten nieder.

Aber Videospiele können sich diese langen Berechnungszyklen nicht leisten. Eine aus Einzelbildern zusammengesetzte Sequenz muss aus 30 bis 60 Einzelbildern bestehen, um für das menschliche Auge flüssig zu wirken. Da die Software unmittelbar auf die Eingabe des Spielers reagieren soll, können die einzelnen Bilder nicht vorher berechnet werden. Für moderne Spiele-Engines bedeutet dies, dass die Berechnung eines einzelnen fotorealistischen Bildes nicht länger als eine Dreißigstelsekunde beanspruchen darf. Um dieses Ziel zu erreichen, werden komplizierte physikalische Formeln verkürzt und visuelle Darstellungen vereinfacht. Diese Ergebnisse sind durch das Aufkommen von neuen Verfahren oft kaum noch von zeitaufwendigen Offline Renderings zu unterscheiden.

Für die Produktvisualisierung ergibt sich hieraus, neben Zeit- und Kostenersparnissen, ein weiterer Vorteil: Objekte lassen sich nicht nur als Standbild darstellen, sondern, ohne Vorberechnungszeit, von allen Seiten interaktiv betrachten.

Wahl der Software

Für unser vorliegendes Projekt sind zwei Kriterien ausschlaggebend: Ein einfacher und effizienter Workflow sowie die Einbindung moderner Echtzeitrendertechniken. Zum Modellieren verwenden wir Modo, welches sich im Bereich der Visualisierung großer Beliebtheit erfreut und auch als Tool zur Erstellung von Game Assets bestens geeignet ist. Für die Texturierung kommt Photoshop zum Einsatz. Da es sich bei PBR lediglich um ein Konzept handelt, das die Gestaltung von Oberflächen bestimmt, wird keine zusätzliche Spezialsoftware benötigt.

Für das Darstellen unseres Objektes kommt Marmoset Toolbag 2 zum Einsatz (**siehe DP 15:07**). Neben einer schnellen Funktion zum Abspeichern von Standbildern bietet es auch die Möglichkeit, frei drehbare dreidimensionale Objekte unter HTML5, ohne zusätzliche Plug-ins, einzu-

binden. Ein Beispiel hierzu findet sich unter **www.patrick-moechel.net/dp.htm**.

Erstellen der Geometrie in Modo

Einzelne Objektinstanzen werden in Modo „Mesh" genannt. Hierbei handelt es sich zunächst um einen leeren Container für Geometrie. Dieser kann beliebig gefüllt werden, etwa mit mehreren Primitives oder selbst erstellter Geometrie.

Ein großer Vorteil dieser Art der Szeneorganisation besteht darin, einzelne oder mehrere Faces sowie ganze Körper, genannt Items, zwischen den Containern per Copy-and-paste hin- und herbewegen zu können. Dementsprechend leicht fällt es, die Szene nach Baugruppen zu sortieren (Canopy, Battery, Servo, Washplate, etc.; **siehe Abb. 1**). Mehrere Mesh Container lassen sich zwecks der Übersicht auch zu Gruppen zusammenfassen. Dieses Szenensetup ermöglicht auch die schnelle Erstellung einer Explosionszeichnung (**siehe Abb. 2**).

Den einzelnen Bauteilen weisen wir verschiedene Farbwerte zu. Nun sortieren wir jedoch nicht nach Baugruppen, sondern nach Materialien. So können einem Teil mehrere Farben zugewiesen werden, wie in **Abb. 3** zu sehen. Da wir diese Farbinformationen als sogenannte Color Map verwenden, sollten sich die verschiedenen Farben möglichst stark voneinander abheben. So fällt es später leichter, den Überblick zu bewahren. Zum Erstellen der Color Map nutzen wir einen „Render Material to Texture" genannten Prozess: Im Shading Tab: Add Layer > Image Map > (new image); Rechtsklick auf das neue Image: „Bake to Texture".

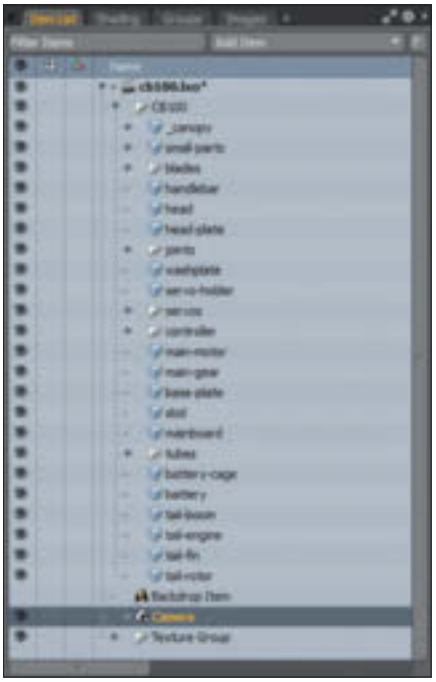

Abb. 1: Jede Baugruppe erhält ihren eigenen Mesh Container.

Diese Technik bietet sich auch hervorragend für das Erstellen von Decal-Masken an. Komplexe Linienverläufe können direkt in die Geometrie geschnitten werden, um so Farbflächen per Edge Loops voneinander zu differenzieren, wie in **Abb. 4** zu sehen ist. Die hieraus gewonnene Textur kann nun als Grundlage zur Erstellung der Decals in Photoshop verwendet werden. Bevor wir mit dem Texturieren beginnen, exportieren wir unser Mesh als .obj-Datei.

Einführung in PBR

Wie am Anfang erwähnt, handelt es sich bei PBR um eine Rendermethode, welche auf stark vereinfachten physikalischen Regeln basiert. Zugrunde liegt die Idee, die jeweiligen Oberflächen mit Informationen über Materialeigenschaften zu versehen. Diese wiederum fließen in die physikalische Berechnung der Lichtbrechung und -reflexion auf der Oberfläche ein.

Abb. 2: Explosionszeichnung mit farblicher Unterscheidung der einzelnen Materialien

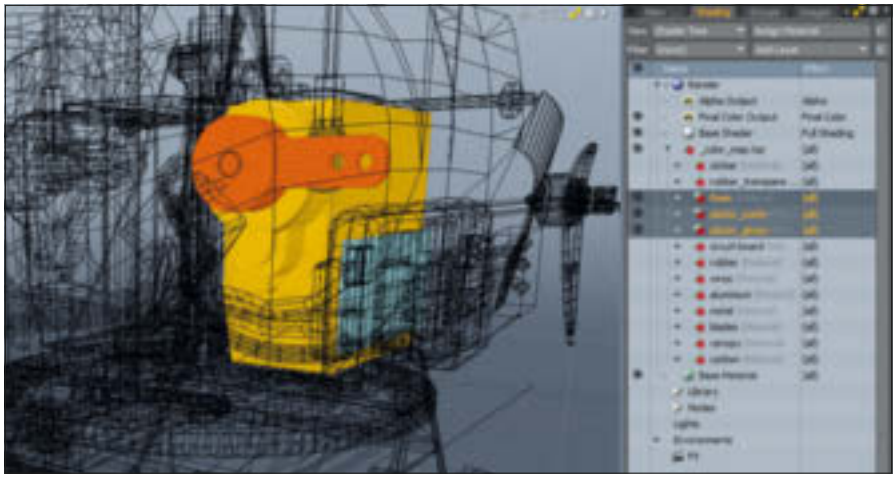

Abb. 3: Materialien werden durch verschiedene Farben deutlich voneinander differenziert.

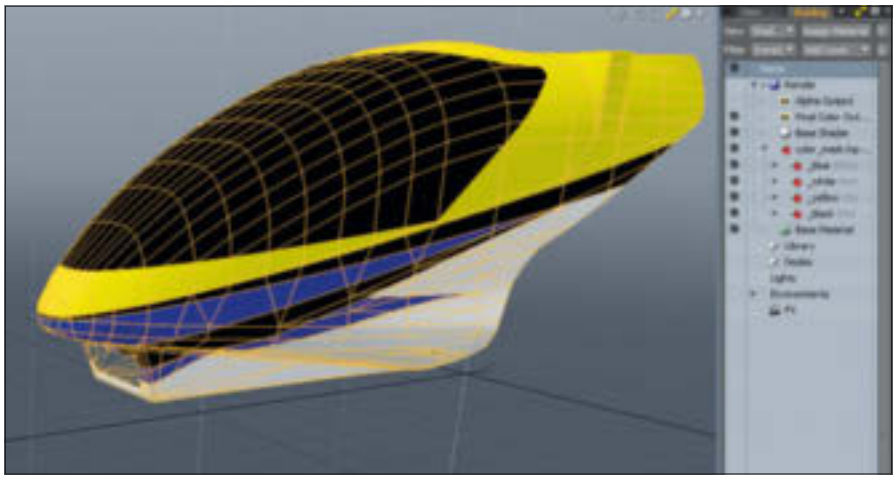

Abb. 4: Flächen müssen nicht als Vektorgrafiken erstellt, sondern können auch direkt in das Mesh gecuttet werden.

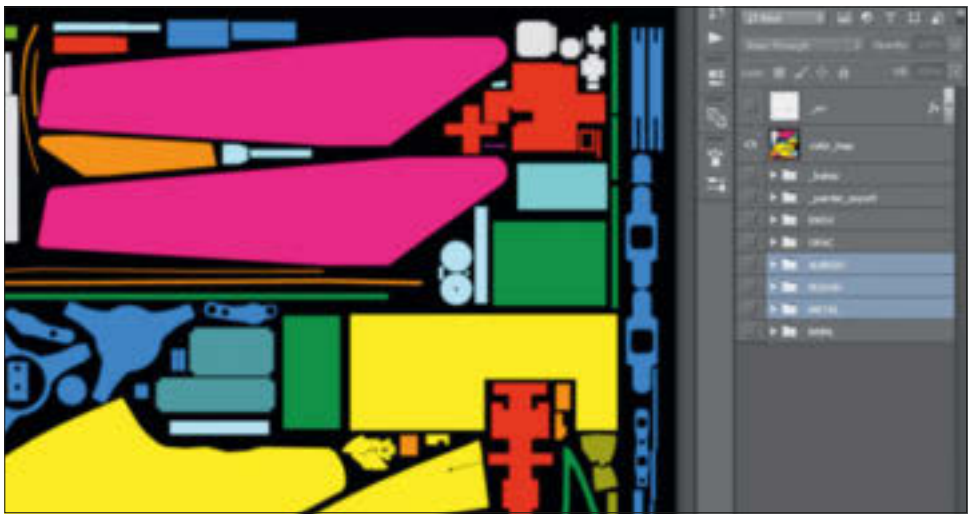

Abb. 5: Die Color Map als Auswahlgrundlage in Photoshop

Abb. 6: Für die Verwendung von Transparenzen muss die Opacity Map im Alphakanal der Albedo Map gespeichert werden.

Abb. 7: Kleine geometrische Details – dargestellt per Normal Map.

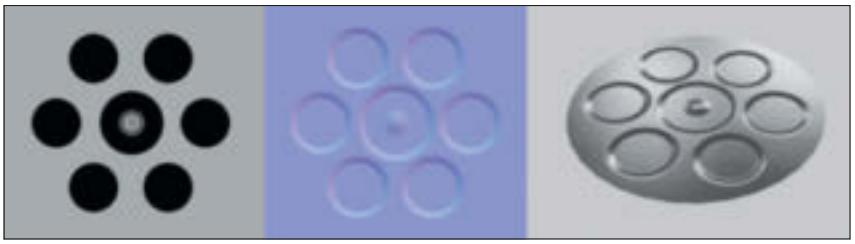

Abb. 8: Vektorform in Photoshop, Normal Map und Echtzeitrender

Hierzu werden folgende Texture Maps benötigt: Albedo (oder Base Color, auch bekannt als Diffuse Map) zur Farbgebung, Metalness für den Grad der Reflexion und Roughness für die Oberflächenstruktur (weiche Materialien, wie zum Beispiel Gummi, haben andere Lichtbrechungseigenschaften wie harte Oberflächen). Metalness und Roughness Maps lassen sich auch als Nachfolger der Specular Map verstehen. Da es sich bei PBR nicht um ein festgelegtes Regelwerk handelt, gibt es in der Praxis Rendermodelle, die mit im Labor gemessenen Werten der Oberflächenbeschaffenheit arbeiten, sowie Renderer, welche auf geschätzten Annäherungswerten basieren. Eine gute Übersicht über entsprechende Werte hat Sébastien Lagarde auf seiner Seite zusammengestellt: **tinyurl.com/nvys3cs**.

Bei klassischen Diffuse- und Specular- respektive Glossiness-Rendermodellen mussten die Grauwerte in den Texture Maps oft noch für spezifische Beleuchtungssituationen von Hand angepasst werden. Dies entfällt bei Physically Based Rendering, da anstelle des Definierens von Werten für die Glanzeigenschaften einer Oberfläche die tatsächlichen Materialeigenschaften vorgegeben werden. Da die Glanzreflexionen nun nicht mehr fest definiert sind, lassen sie sich im Rahmen von PBR auf Basis der physikalischen Attribute einer Oberfläche in Echtzeit berechnen und adäquat darstellen.

Texturing in Photoshop

Für die Erstellung der Texture Maps öffnen wir zunächst ein neues Photoshop-Dokument. Hier bekommt jede Texture Map ihren eignen Ordner (**Abb. 5**). Weitere mögliche Texture Maps sind: Emissive für Glow-Effekte, Opacity für Transparenz und Normal Maps für die Simulation von geometrischen Oberflächenstrukturen – hierzu später mehr.

Zusammen mit der zuvor gerenderten Color Map und dem Magic Wand Tool in Photoshop können wir nun zügig Masken erstellen und Grauwerte für die einzelnen Materialien in unserer Roughness und Metalness Map festlegen.

What you see is what you get

Für die Feineinstellung der jeweiligen PBR-Werte nehmen wir uns Marmoset Toolbag 2 zur Hilfe. Hierbei handelt es sich um ein schlankes Tool zur PBR-Echtzeitberechnung von 3D-Modellen. Besonders angenehm ist die einfache Bedienung. Im Gegensatz zu vollwertigen Game Engines müssen keine Szenen erstellt werden – Umgebungen und Lichteinstellungen sind vordefiniert und einfach auszuwählen. Für individuelle Anpassungen können auch eigene Setups erstellt werden. Meshes und Texture Maps lassen sich einfach per Drag-and-drop hinzufügen und Objekte in Echtzeit im 3D-Raum betrachten.

Marmoset Toolbag 2 Workflow

Durch die Kombination von Photoshop und Marmoset Toolbag 2 in unserem Arbeitsablauf erhalten wir zwei Vorzüge auf einmal: Die umfassende Funktionalität und Flexibilität von Photoshop und die Möglichkeit, unsere Texturen in PBR und Echtzeit zu sehen. Hierzu genügt ein einfacher Programmwechsel mit Alt+Tab nach dem jeweiligen Speichern einer Textur. Marmoset Toolbag 2 lädt veränderte Dateien, nachdem sie gespeichert wurden, automatisch neu.

Um eine neue Szene einzurichten, ziehen wir als Erstes unsere zuvor exportierte *.obj-Datei in den Viewport. Klicken wir nun auf „Sky“ im Scenebrowser, können wir unter „Presets“ diverse Lichtstimmungen

laden. Falls auf das Hintergrundbild verzichtet werden soll, stellen wir den Backdrop Mode von „Sky“ auf „Color“ respektive „Ambient Sky“ für einen Gradient im Hintergrund. Texturen werden geladen, indem man erst auf das Default Material und dann auf das leere Fenster im entsprechenden Texture Slot klickt. Die Einstellungen für unser Material sind in **Abb. 6** zu sehen.

Geometrische Details zeichnen statt modellieren

Eine in der Videospielgrafik beliebte Methode, um Performance zu sparen, sind sogenannte Normal Maps. Für das Sparen von Rechenleistung werden geometrische Feinheiten bewusst nicht modelliert, um somit weniger Details im Mesh berechnen zu müssen. Stattdessen wird eine zusätzliche Textur verwendet. Hier werden Informationen über die Abweichung von der Oberfläche des Meshs per Pixel gespeichert (**siehe Abb. 7**). Vereinfacht beschrieben steht jeder Farbton in der Normal Map für eine bestimmte Winkelgradzahl, um die Differenz zu der Normalen des darunterliegenden Face zu beschreiben.

Ein gängiger Workflow beim Erstellen von Normal Maps ist das Projizieren von Highpoly Meshes auf Modelle mit einem niedrigeren Detailgrad (Baking). Dieser Prozess ist vergleichbar mit dem beschriebenen Verfahren, Farbinformationen aus dem Shader in eine Textur zu rendern – nur handelt es sich hier nicht um Farb-, sondern eben um Geometrieinformationen.

Eine weitere Möglichkeit Normal Maps zu erstellen, besteht darin, aus Graustufenbildern Höheninformationen abzuleiten und diese in Normalinformationen zu konvertieren. Praktisch bedeutet dies, dass Vertiefungen gegen Schwarz und Erhöhungen gegen Weiß eingezeichnet und im nächsten Schritt per Algorithmus in Normalinformation umgewandelt werden. In Photoshop befindet sich diese Funktionalität unter: Filter > 3D > Generate Normal Map … (**vgl. Abb. 8**).

Benutzer älterer Photoshopversionen können sich ein kostenloses Plug-in von der Nvidia-Developer-Seite herunterladen: **tinyurl.com/paz97yz**.

Abb. 9: Gängige Einstellungen wie Color Correction, DOF und stereoskopische Darstellung sind bereits enthalten.

Finaler Render

Marmoset Toolbag 2 enthält alle auch in geläufigen Game Engines gängigen Einstellungen wie zum Beispiel Color Correction oder Depth of Field (**siehe Abb. 9**). Zusätzlich verfügt das Tool über die Möglichkeit der stereoskopischen Darstellung, unter anderem mit dem VR Headset Oculus Rift DK1 (Render > Stereo 3D > Stereo Mode).

Des Weiteren bietet das Programm die Möglichkeit, eine Szene zur Darstellung auf einer Webseite zu exportieren. Dabei wird eine .mview-Datei erstellt (File > Viewer Export …), welche ohne weitere benötigte Browser-Plug-ins direkt in HTML5 eingebunden werden kann:

```
<head>
<script src=“https://viewer.marmoset.co/
main/marmoset.js“></script>
</head>

<body>
<script>
marmoset.embed( ‚scene.mview‘, { width:
800, height: 600, autoStart: true, full-
Frame: false, pagePreset: false } );
</script>
</body>
```

Breite und Höhe des eingebetteten Viewports werden mit den Werten „Width“ und „Height“ definiert. Es gilt zu beachten, dass eine Vorschau im Browser nicht mit lokalen Quelldateien, sondern nur von einem Webserver aus funktioniert.

Fazit

Der beschriebene Workflow ermöglicht nicht nur die qualitativ hochwertige Darstellung in Echtzeit, sondern auch, Änderungen im Modell und in der Textur sofort sichtbar zu machen. Es entstehen keine Wartezeiten durch aufwendige Renderprozesse. So werden Kosten- und Zeitaufwand bei Iterationen minimiert. Die Darstellung ist nicht auf Standbilder beschränkt. Gegenstände lassen sich von allen Seiten in beliebigen Beleuchtungssituationen betrachten.

Zusätzlich bietet unser Projektaufbau die Grundlage für den weiteren Gebrauch in qualitativ und technologisch ausgereiften Echtzeitumgebungen wie der Unreal Engine.

Animationen wie Kamerafahrten oder Turntables sowie Simulationen, wie Time of Day für zum Beispiel Architekturvisualisierung oder auch Wettereffekte, können hier schnell und effektiv realisiert werden. Custom Interfaces für interaktive Produktdemos sind ebenfalls denkbar. **› ei**

Patrick Möchel ist Head Instructor für den Bereich Game Art am SAE Institute Hamburg. Zuvor arbeitete er als Environment Artist für Sony bei Guerrilla Games Amsterdam an den PS4-Titeln „Killzone: Shadow Fall“ und „Horizon: Zero Dawn“ sowie für Crytek in FFM. Hier trug er als 3D-Artist zu „Crysis 2“ (PC, Xbox 360, PS3) bei.