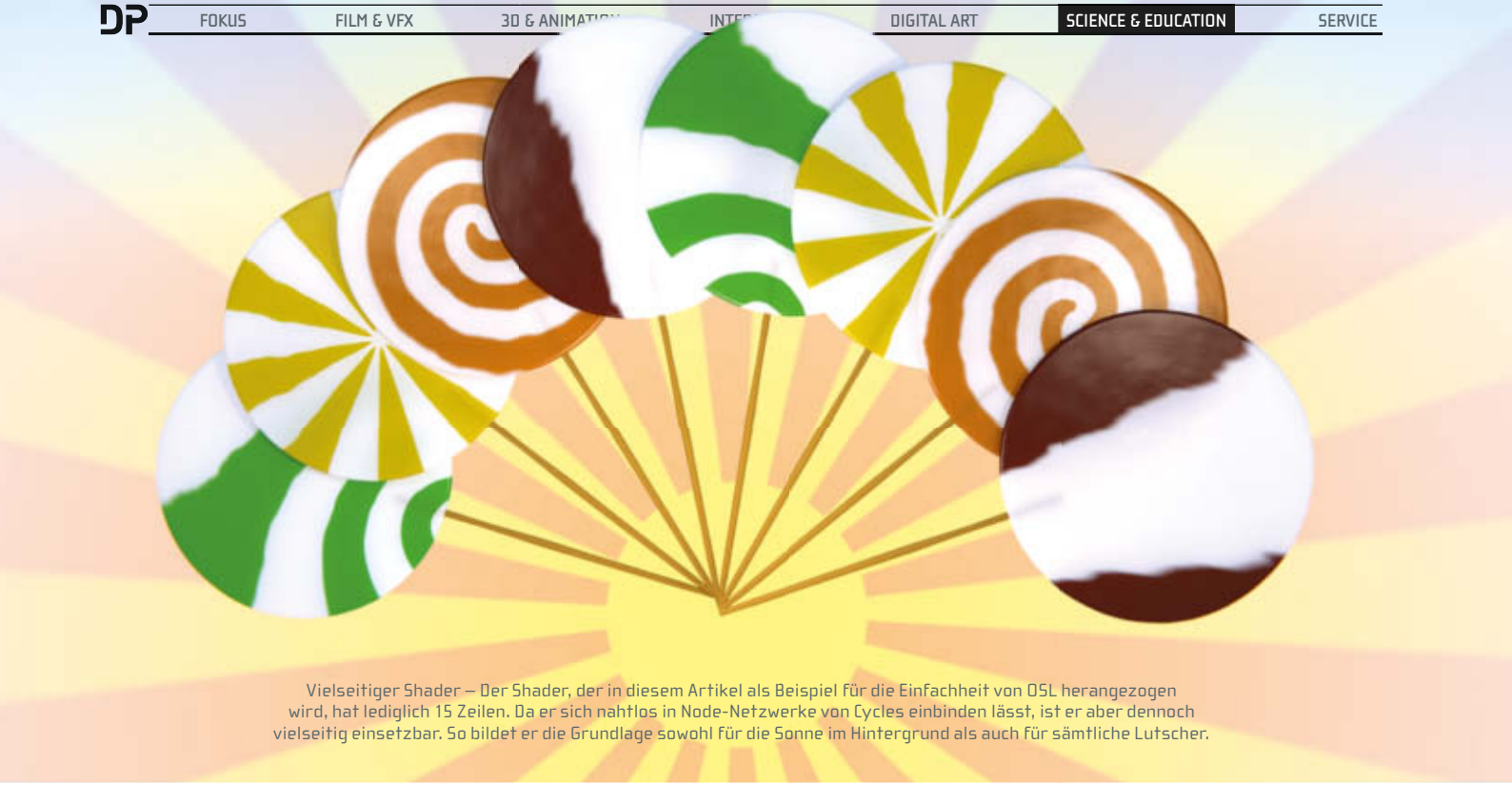


Vielseitiger Shader – Der Shader, der in diesem Artikel als Beispiel für die Einfachheit von OSL herangezogen wird, hat lediglich 15 Zeilen. Da er sich nahtlos in Node-Netzwerke von Cycles einbinden lässt, ist er aber dennoch vielseitig einsetzbar. So bildet er die Grundlage sowohl für die Sonne im Hintergrund als auch für sämtliche Lutscher.

Open Shading Language (OSL) in Blender Cycles

Für Renderman gibt es die Renderman Shading Language (RSL), für OpenGL die OpenGL Shading Language (GLSL) und für DirectX die High-level Shading Language (HLSL). Mit der Open Shading Language (OSL) hat sich nun eine weitere dazugesellt. Diese ist für moderne Pathtracer ausgelegt und wurde im Arnold Renderer bereits ausgiebig genutzt, unter anderem für die Filme „Men in Black 3“, „The Amazing Spider-Man“ und „Die fantastische Welt von Oz“. von Gottfried Hofmann

OSL soll in Zukunft in diverse Render-Engines Einzug halten, unter anderem in V-Ray. Daher macht es Sinn, sich die Sprache schon einmal anzusehen. Da trifft es sich gut, dass die Render-Engine „Cycles“ von Blender mit OSL-Unterstützung daherkommt.

Shader-Sprachen und ihre Vorteile

Wer Erfahrung mit Renderman oder OpenGL hat, wird die Flexibilität und Schnelligkeit beim Bau komplexer Shader zu schätzen wissen. Vor allem wenn größere mathematische Formeln ins Spiel kommen, eignet sich eine auf Text basierte Programmiersprache einfach besser als das langwierige Zusammenklicken von Nodes. Spätestens wenn man sich ein fremdes Setup ansieht, wird das klar. Denn vom Aussehen einer mathematischen Formel bleibt bei Nodes meist nicht viel übrig. Vor allem wenn so Dinge ins Spiel kommen, wie die Reihenfolge der Ausführung oder wenn man so einfache mathematische Dinge wie Klammern nur durch Tricks erreichen kann. Die Notation in einer Shader-Sprache ist da einfach viel näher am „Original“, sprich der Formel, wie man sie auch auf Papier notieren würde. OSL zum Beispiel orientiert sich in der Syntax an C. Auch können die Ergebnisse deutlich übersichtlicher ausfallen als riesige Node-Setups.

Der größte Vorteil einer Shader-Sprache ist aber immer noch die einfache Austauschbarkeit zwischen Applikationen. Momentan können auf dem Markt für Endkunden nur die Render-Engine „Cycles“ von Blender und der Szenen-Beleuchtungs-Editor für Computerspiele „Beast“ von Autodesk mit OSL-Unterstützung aufwarten. Mitte vergangengen Jahres hat aber der Hersteller Chaos Group angekündigt, in zukünftigen Versionen von V-Ray ebenfalls OSL zu unterstützen. Damit vereinfacht sich der Austausch von Shadern immens. Ein in V-Ray oder Cycles erstellter OSL-Shader ließe sich dann einfach als Text-Datei in den jeweils anderen Renderer laden. Da sich in OSL auch prozedurale Texturen implementieren lassen, kann man OSL getrost auch als plattform- und renderübergreifendes Plug-in-System verstehen. Einige Noise-Arten, darunter auch das noch relativ junge Gabor-Noise, sind bereits in der Basis-Installation von OSL vorhanden, während sich zahlreiche weitere als Skripte im Internet finden.

Nähe zu Renderman und GLSL sowie Unterschiede

Die Syntax sowie die vorhandenen Funktionen von OSL zeigen eine sehr große Nähe zur Shader-Sprache von Renderman (RSL), wodurch sich in dieser Sprache geschriebene Shader äußerst einfach zu OSL konvertieren lassen. Im Internet finden sich eine Fülle von Renderman-Shadern zum Adaptieren. Zum Erlernen von OSL eignen sich neben Einsteiger-Tutorials somit auch Bücher über Renderman. Ähnlich einfach gestaltet sich die Portierung von GLSL-Shadern. Dabei muss allerdings beachtet werden, dass OSL für moderne Pathtra-

Portierter Renderman-Shader – Diese prozedurale Parkett-Textur wurde zuerst im Jahr 1995 von Larry Gritz für Renderman entwickelt. 2012 konnte sie mit nur minimalen Änderungen von Brecht van Lommel zur Open Shading Language konvertiert werden.

cer konzipiert wurde und dadurch ein paar Einschränkungen hat. Ausgegeben werden zum Beispiel sogenannte Clusores, die von der Render-Engine interpretiert werden. Auf deren Werte kann man nicht direkt zugreifen, während bei anderen Skriptsprachen für die Oberfläche RGB-Werte zurückgegeben werden und man die Helligkeit auf einer Oberfläche direkt als Maske nutzen kann. Bei OSL hingegen arbeitet man ausschließlich mit dem einfallenden Licht und wie das Material dieses verändert.

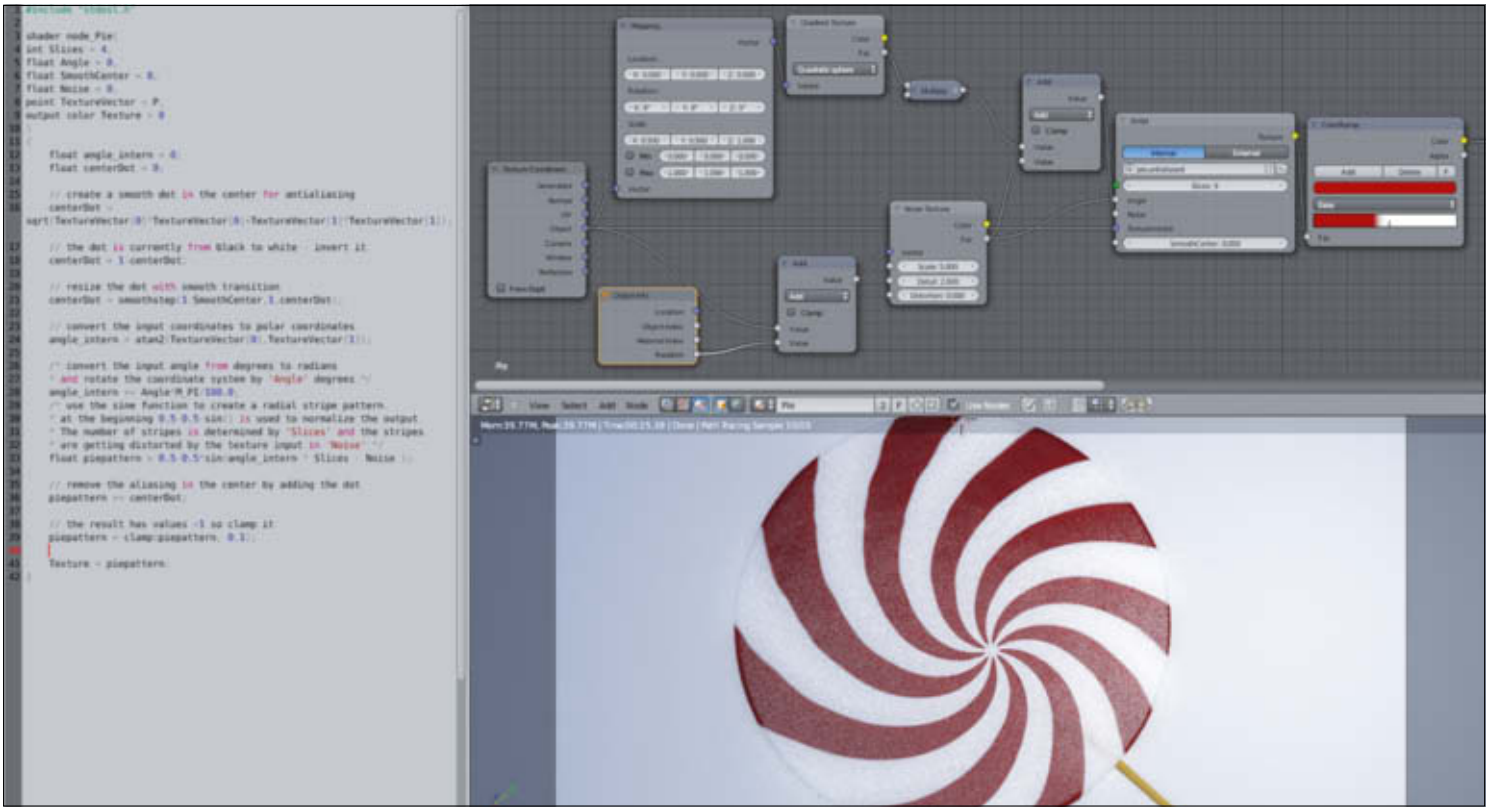
Der Vorteil ist, dass man Probleme wie Energieerhaltung bis auf Sonderfälle nicht mehr im Shader behandeln muss, was vor allem für fotorealistische Renderings einige Dinge einfacher gestaltet. Wer Erfahrung mit Blender Cycles gesammelt hat, ist mit dem Konzept bereits vertraut.

Nur ein Werkzeug für TDs?

Die Programmierung von Shadern ist zwar nicht jedermanns Sache, aber OSL hat gegenüber anderen Shader-Sprachen den eindeutigen Vorteil, dass die Shader für die Anwendung in einem Node-Netzwerk konzipiert sind. Sprich die Eingabe kann von einem anderen Shader stammen und die Ausgabe von weiteren Shadern genutzt werden. Für den Nutzer, der für seine Aufgabe passende Shader aus dem Internet bezieht, erscheint OSL, wie schon angesprochen, als einfaches Plug-in-System, mit dem sich die Fähigkeiten des eingesetzten Renderers gezielt erweitern lassen können. Die Vorteile für Programmierer wie für Anwender sollen an folgendem, einfachen Beispiel aufgezeigt werden.

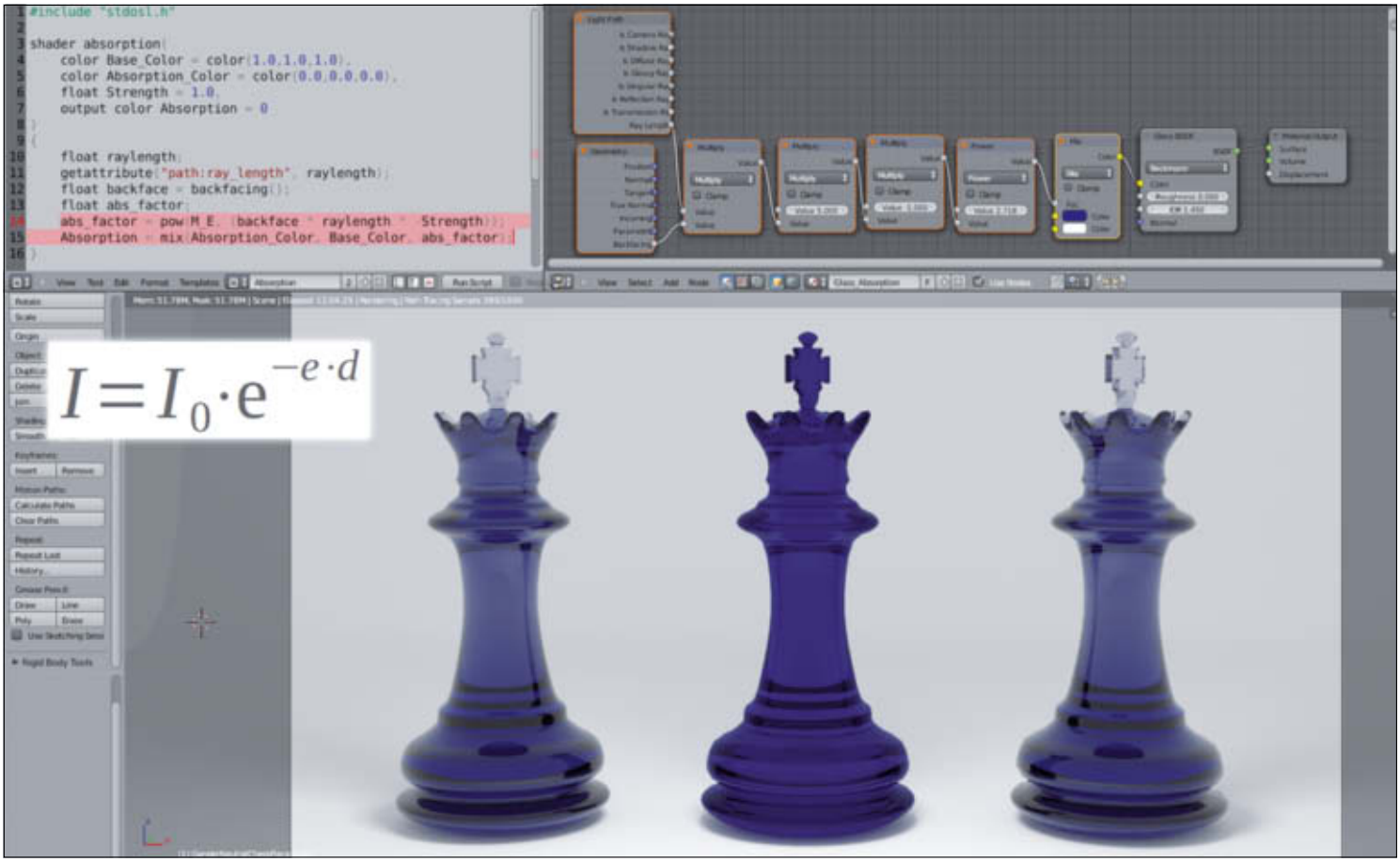
Die alten Textur-Plug-ins von Blender vs. OSL

Blender 2.49 hatte bereits ein Plug-in-System für prozedurale Texturen etc., das allerdings für Programmierer gegenüber OSL einiges an Mehraufwand bedeutete. So musste für jedes Betriebssystem, auf dem Blender läuft, ein eigenes Plug-in kompiliert werden. OSL erledigt das automatisch zur Laufzeit. Außerdem musste auf sehr niedrigem C-Level gearbeitet werden, wodurch sich für den Programmierer nochmals mehr Aufwand einstellte. Der folgende Code erzeugt eine einfache, radiale Textur ähnlich den Stücken von Kuchen bzw. Strahlen:



OSL im Node-Netzwerk – Diese prozedurale Textur erzeugt ein radiales Muster von Elementen ähnlich einem Tortendiagramm. Sie kann direkt im Shader-Netzwerk genutzt werden. Dadurch können viele Sachen, die man eigentlich im Shader-Code definieren müsste, über zusätzliche Nodes erledigt werden. Die Farb-Rampe auf der rechten Seite kümmert sich nicht nur um die Farbe, sondern auch um die Schärfe. Auf der linken Seite finden sich diverse andere prozedurale Texturen, die dazu dienen, das Muster in sich zu verdrehen und Dellen hinzuzufügen.

```
/* Copy lefted. (GPL v2.0)
 * (source by Robert Wenzlaff,
7/16/00)
 */

#include „math.h“
#include „plugin.h“
#define NR_TYPES        1

float result[8];
float cfra;
int  do_reset = FALSE;
extern float hnoise(float noise-
size, float x, float y, float z);

/* set up plugin menu */

char name[]= „Pie“;
char stnames[NR_TYPES][16]= {„Pie“};

VarStruct varstr[]= {
   /* type,      name,
default,min,     max,     tooltips */
   { LABEL,      „“,      0,       0,
0, „“},
   { NUM|INT,    „divs „,          6.0,
2.0,  1000.0,
   „Pie plugin: Number of slices“},
        /* Number of slices */
   { NUM|FLO,    „hardness „, 1.0,
0.0,  5.0,
   „Pie plugin: Falloff Hardness“},
        /* determins sharpnes of
edge, could use some work */
   { NUM|FLO,    „ang ofs „, 0.0,
-180.0, 180.0,
   „Pie plugin: Angle offset „},
        /* phase angle */
   { NUM|FLO,    „turb dep „,0.0,
-5.0,    5.0,
   „Pie plugin: Turbulance depth“},
        /* Pos. Turb affects ‚white‘,
Neg. affects ‚black‘ */
   { NUM|FLO,    „turb siz „,0.25,
0.0,  2.0,
   „Pie plugin: Turbulance size“}
};
typedef struct Cast {
   float dum1;
   int div;
   float hard;
   float ang;
   float turbd;
   float turbs;
} Cast;

/* ********************************/

int plugin_tex_doit(int, Cast*,
float*, float*, float*);

int plugin_tex_getversion(void)
{
   return B_PLUGIN_VERSION;
}

void plugin_but_changed(int but)
{
}

void plugin_init(void)
{
}

void plugin_getinfo(PluginInfo
*info)
{
   info->name= name;
   info->stypes= NR_TYPES;
   info->nvars= sizeof(varstr)/
sizeof(VarStruct);

   info->snames= stnames[0];
   info->result= result;
   info->cfra= &cfra;
   info->varstr= varstr;

   info->init= plugin_init;
   info->tex_doit= (TexDoit) plug-
in_tex_doit;
   info->callback= plugin_but_chan-
ged;
}

int plugin_tex_doit(int stype, Cast
*cast, float *texvec, float *dxt,
float *dyt)
{
   float angle, turb=0;

   angle = atan2(texvec[0],texvec[1])
+ cast->ang*3.1415926/180.0;

   if ( cast->turbd !=0.0 ) { /* save
time if no turb */
      turb = cast-
>turbd * hnoise(cast-
>turbs,texvec[0],texvec[1], tex-
vec[2]);
      /*printf(„turb: %f\n“,turb);*/
   }

   result[0]= 0.5 - 0.5*sin(angle *
cast->div - turb - 0.5);
   if (cast->hard !=1) result[0]=
```



Code vs. Nodes – Ein einfacher Shader für volumetrische Absorption, basierend auf dem Lambert-Beerschen Gesetz, allerdings abgewandelt für den künstlerischen Gebrauch. Links oben der Shader als OSL-Code, rechts oben als Node-Setup in Cycles. Die linke (gender-neutrale) Schachfigur ist mit dem OSL-Shader ausgestattet, die rechte mit dem normalen Cycles-Setup. In der Mitte zum Vergleich mit normalem Cycles-Glas. Zum Vergleich noch die Formel, auf der der Effekt basiert.

```
pow(result[0], cast->hard); /*Very
slow, better way?*/

 if (result[0] > 1) result[0] = 1.0;
   else if (result[0]<0) result[0]=0;
   result[4]= 1.0;
   return 0;
}
```

Die OSL-Version kommt im Vergleich dazu mit deutlich weniger Code-Zeilen aus und ist übersichtlicher zu lesen:

```
    #include „stdosl.h“

    shader node_Pie(
    int Slices = 4,
    float Angle = 0,
    float Noise = 0,
    point TextureVector = P,
    output color Texture = 0
    )
    {
     float angle_intern = 0;
     angle_intern = atan2(TextureVector[0],Tex
tureVector[1]) + Angle*M_PI/180.0;
     Texture = 0.5-0.5*sin(angle_intern * Sli-
ces - Noise);
    }
```

Obendrein ist sie auch noch vielseitiger einsetzbar. So war in der ursprünglichen Textur eine Noise- respektive Turbulence-Textur definiert, die dafür genutzt werden konnte, die einzelnen Strahlen zu verzerren oder zu verwirbeln. Aber warum sollte man sich dafür auf eine einzelne Textur festlegen? In der OSL-Version wird für besagte Textur einfach ein Input definiert. Daran kann jede beliebige Textur angeschlossen werden, also zum Beispiel auch ein Bild. Und das Bild wird nicht einfach nur auf die Textur gelegt, sondern verzerrt sie. Wie das aussehen kann, zeigt das Intro-Bild dieses Artikels. Außerdem war in der ursprünglichen Textur ein Parameter vorhanden, um die Schärfe der Strahlen zu verändern. Diese Aufgabe könnte aber eine Color-Ramp in Cycles viel besser erledigen. Daher wurde in der OSL-Version auf diesen Parameter verzichtet.

Angesicht der Tatsache, dass das Schreiben eines OSL-Shaders mit viel weniger Aufwand verbunden ist, als das Erstellen eines Plug-ins für das alte System, ist es wenig verwunderlich, dass wenige Wochen nach der Implementierung von OSL in Blender bereits mehr brauchbare Shader verfügbar waren, als für das alte Plug-in-System in mehr als zehn Jahren jemals geschrieben worden sind. Mit der Open Shading Language hat die 3D-Welt ein spannendes neues Plug-in-System erreicht, das den Austausch zwischen den Programmen hoffentlich erleichtern wird. Entwickler können dabei auf einen reichhaltigen Erfahrungsschatz zurückgreifen, die mit den bereits vorhandenen Shader-Sprachen gemacht wurden.

Interview mit Larry Gritz

Larry Gritz ist Hauptentwickler der Open Shading Language bei Sony Pictures Imageworks. Ein weiteres bekanntes Open-Source-Projekt von ihm ist OpenImageIO. Larry Gritz ist zudem Co-Autor des Buchs „Advanced RenderMan: Creating CGI for Motion Pictures“. Viele bekannte Beispiel-Shader für Renderman gehen auf ihn zurück.

DP: **Wie hat sich der Shading-Workflow bei Sony Pictures Imageworks (SPI) mit der Einführung von OSL geändert?**
Larry Gritz: Bevor wir OSL hatten, waren die Shader bei SPI kompilierte C-Plug-ins für den Arnold Renderer. Unsere Shader-Entwickler mussten immer mehr Zeit in Low-Level-Details investieren. Zum Schreiben von Shadern ist reines C etwas klobrig und fehleranfällig. Auch die Kopplung an die Render-Interna sorgte immer wieder für Kopfschmerzen. Außerdem wurde es immer schwieriger, bei Materialien und Lichtern den Grad an physikalischer Korrektheit zu erlangen, den wir haben wollten. Es war irgendwann klar, dass wir eine andere Form der Abstraktion brauchten, als was in unserem bisherigen Shader-System genutzt wurde.
OSL hat für uns gleich drei Dinge auf einmal geändert: Erstens sind Syntax und Semantik der Sprache auf das Schreiben von Shadern ausgelegt, es ist einfach eine viel angenehmere Notation, um das Shading eines Materials zu beschreiben, vor allem wenn man es mit rohem C-Code vergleicht. Zweitens präsentiert OSL dem Shader-Entwickler eine saubere API, die hässliche oder komplexe Render-Interna nicht direkt aufzeigt. Diese beiden Dinge zusammen bedeuten, dass fast alle Code-Zeilen eines Shaders – und damit auch die Aufmerksamkeit des Entwicklers – sich auf die Beschreibung des Materials konzentrieren. Viel Arbeit, die aus dem Herumschieben von Daten und Auseinandersetzungen mit Render-Interna bestand, wurde dadurch eliminiert.

Zu guter Letzt finden momentan Veränderungen beim Design der Renderer statt. Für die Beschreibung des Aussehens eines Materials gab es bisher nur eine Sicht, was bedeutet, dass all die unschönen Details aus dem Bereich Sampling und Integration im Shader selbst abgelegt sind. Stattdessen wird heutzutage verstärkt auf die Berechnung von Closures gesetzt. Diese überlassen das Sampling und die Integration der Render-Engine, die normalerweise viel bessere Voraussetzungen mitbringt, um die Aufgabe gut zu erledigen. Diese Veränderungen haben uns sehr bei unseren Bemühungen geholfen, unsere Modelle für Licht und Materialien physikalisch noch korrekter zu gestalten und naturgetreue Bilder „out of the box“ zu erhalten.

DP: **Was war die Motivation, OSL als Open Source zur Verfügung zu stellen?**
Larry Gritz: Proprietäre Werkzeuge können manchmal einen Vorteil gegenüber der Konkurrenz sichern, verursachen aber auch reale Kosten: Jedes Tool und Format, das nur in deinem Studio eingesetzt wird, ist auch eines, für das du niemanden anheuern kannst, der schon ein Experte ist. Du wirst nie externe Dokumentationen finden und deine von Drittanbietern zugekaufte Software wird sie nicht unterstützen, ohne dass du selbst die Plug-ins schreiben musst. Wir haben also abgewogen und schätzen, dass es zwar ein Vorteil für uns wäre, wenn wir OSL behalten würden, wir aber deutlich mehr zurückerhalten, wenn es ein Ökosystem rund um OSL gäbe, das über SPI hinausreicht. Wenn zum Beispiel die Software von Drittanbietern irgendwann OSL unterstützt, könnten wir Technical Directors (TDs) einstellen, die sich mit dieser Methode des Shadings schon auskennen. Ein OSL-Ökosystem wäre von uns sogar dann von Vorteil, wenn die Leute, die wir einstellen, noch keine Erfahrungen damit haben. Denn Künstler wechseln von Studio zu Studio und es ist attraktiver für einen TD, Fähigkeiten zu erlernen, mit denen er etwas anfangen kann, wenn er SPI verlässt, als sich mit Details herumzuschlagen, die er außerhalb nie wieder sehen wird.

Außerdem ist es von professioneller Seite her eine große Genugtuung für die Software-Entwickler. Sie bekommen häufig keine Credits im Abspann der Filme und können nur schlecht bestimmte Shots präsentieren, an denen sie mitgearbeitet haben. Also Dinge, die von außen sichtbar sind und Anerkennung von Kollegen von anderen VFX-Studios bringen. Wenn du weißt, dass andere deine Arbeiten tatsächlich sehen, gibst du dir mehr Mühe, soliden Code zu schreiben, auf den du stolz sein kannst. Wir glauben tatsächlich, dass die Implementierung von OSL besser ist, weil es ein Open-Source-Projekt ist.

DP: **Habt ihr schon Beiträge von außen bekommen?**
Larry Gritz: Ja! Selbst als OSL noch nicht implementiert war, haben wir frühe Design-Dokumente an andere Studios geschickt und bekamen sehr gutes Feedback, selbst von anderen bekannten Studios. Wenn es um tatsächliche Code-Beiträge geht, die Hauptentwickler arbeiten bei SPI, aber wir hatten viele Beiträge von anderswo. Besonders seit OSL jetzt in Blender/Cycles, V-Ray und Autodesk Beast eingebaut wird, bekommen wir regelmäßig Beiträge von den Entwicklern dieser Pakete. Dazu gehören kleinere Bugfixes, Optimierungen und Änderungen, die beim Portieren und Kompilieren auf anderen Plattformen helfen, die wir selbst nicht einsetzen. Wir haben zum Beispiel viel Hilfe erhalten, damit OSL gut auf Windows läuft.

» Es ist nicht einfach, auf ein anderes Shading hinzuarbeiten. «

Larry Gritz
Hauptentwickler Sony Pictures Imageworks

DP: **OSL wurde im Jahr 2010 als Open Source herausgebracht, jetzt haben wir 2013 und Blender Cycles ist immer noch die einzige Render-Engine auf dem Markt, die OSL unterstützt. Sind die Entwickler anderer Render-Engines zögerlich bei der Adaption und falls ja, was könnten die Gründe sein?**
Larry Gritz: So ganz stimmt das nicht. OSL steckt auch in Autodesk Beast und wird derzeit in V-Ray integriert. Beides Produkte mit breiter Nutzerbasis. Außerdem, obwohl OSL seit 2010 Open Source ist in dem Sinne, dass wir viel der anfänglichen Entwicklung im Offenen durchführten, hat es bis Mitte 2012 gebraucht, dass SPI die ersten Filme fertigstellte, bei denen ausschließlich OSL für das Shading genutzt wurde. Das war ein wichtiger Meilenstein, auf den viele Leute warteten, bevor sie auf OSL setzen wollten. Sprich, eigentlich ist es erst ein Jahr her, seitdem wir so etwas wie eine Version 1.0 in einem stabilen Status haben, die sich in diversen Produktionen bewiesen hat. Seither wissen wir von drei wichtigen Engines, die OSL integriert haben.

Es ist außerdem schwer, eine existierende Render-Engine auf OSL zu portieren, aus drei Gründen:

1. Die Shader Execution Engine (und mit großer Sicherheit auch das Textur-System) ist ein sehr großer Teil eines Renderers. Der müsste durch ein externes Paket ersetzt werden, über das man nicht zwingend die volle Kontrolle hat.

2. Einen existierenden Renderer zu OSL upzugraden, dürfte auch eine grundlegende Überarbeitung der Bereiche Beleuchtung und Sampling benötigen, selbst in Teilen des Renderers, die von OSL unabhängig sind.

3. Produkte mit einem großen Kundenstamm haben eine große Trägheit, was die Ausbildung der Nutzer, deren Erfahrungen und Shader-Bibliotheken angeht. Es ist also nicht einfach, auf eine andere Art des Shadings hinzuarbeiten.

DP: **Die Syntax von OSL ist sehr ähnlich zur RenderMan Shading Language (RSL). Wurde die so entschieden, damit Nutzer ihre existierenden Renderman-Shader einfach portieren können?**
Larry Gritz: RenderMan Shading Language hat vieles richtig gemacht. Wir wollten darauf aufbauen, wo es funktionierte, aber waren auch nicht zimperlich davon abzuweichen, wo wir Dinge verbessern konnten. Wir haben uns nicht wirklich Gedanken gemacht, dass existierende Shader leicht portiert werden konnten, aber wir wollten auf dem aufbauen, was Shader-Programmierer von anderen Shader-Sprachen her schon kennen. RSL wiederum ist sehr ähnlich zu C, wie auch die meisten der anderen vielgenutzten Shader-Sprachen (GLSL, Gc, Mantra etc.), aus ähnlichen Gründen. Es bringt keine Vorteile, wenn man mit einer Sprach-Syntax anfängt, die sich stark von denen unterscheidet, an die sich Shader-Schreiber gewöhnt haben.

Interview mit Thomas Dinges

Thomas Dinges ist Blender-Entwickler und war eine treibende Kraft hinter der Implementierung von OSL in Blender. Er betreibt außerdem die Webseite OpenShading.com, die mit Informationen, News und Tutorials für OSL in Cycles aufwartet.

DP: **Hallo Thomas, du bist einer der Beteiligten bei der Implementierung der Open Shading Language in Blender Cycles. Was waren bisher deine Erfahrungen dabei?**
Thomas Dinges: Eigentlich haben wir in Cycles bereits seit 2011 eine OSL-Unterstützung gehabt. Damals war OSL zwar schon implementiert, aber noch nicht funktionsfähig. Brecht van Lommel, der Hauptentwickler von Cycles, hat das OSL-Projekt maßgeblich genutzt, um Cycles zu designen. OSL war ein wichtiger Bestandteil des Entwicklungsprozesses von Cycles, man findet heute noch viele Namen und Tools aus OSL, in der Cycles-Engine und im Workflow wieder. OSL ist eines von zwei Shading-Backends, da OSL nur auf der CPU funktioniert und Cycles als Renderer mit GPU-Unterstützung konzipiert war. So kam es, dass Brecht ein weiteres Shading Backend schreiben musste, das auf der GPU läuft. Diesem wurde dann auch sämtliche Aufmerksamkeit gewidmet. OSL erreichte erst im Spätsommer 2012 einen funktionierenden Zustand. Da habe ich ein wenig in den Code geschaut, Kleinigkeiten geändert und Fehler behoben. Später habe ich das Backend mit den Änderungen und neuen Funktionalitäten des GPU-Backends aktualisiert und zusammen mit Lukas Tönne an der Implementierung gearbeitet, so dass wir einen Prototyp zeigen konnten. Dieser hatte auch schon die Script-Node, die von Lukas Tönne erstellt wurde. Damit war es möglich, eigene OSL Shader in Cycles zu nutzen.

» V-Ray wird auf jeden Fall eine OSL-Implementierung haben. «

Thomas Dinges
Blender-Entwickler

DP: **Sprich Blender hatte theoretisch schon seit 2011 eine gewisse OSL-Unterstützung und seit 2012 dann richtig. Inzwischen haben wir aber**

2013 und Blender ist immer noch die einzige Software auf dem freien Markt mit OSL-Shading. Woran könnte es deiner Meinung nach liegen, dass andere Render-Engines so spärlich nachziehen?

Thomas Dinges: Ich denke, das liegt vor allem daran, dass es vor 2012 noch keine Projekte in der Öffentlichkeit gab, die mit Hilfe von OSL realisiert wurden. 2012 kamen „Men in Black III", „Hotel Transylvania" und „The Amazing Spiderman" heraus. Das waren die ersten Feature-Filme, die komplett mit OSL geshaded wurden. Zuvor war OSL noch recht unbekannt. Man wusste zwar, dass es bei Sony Pictures Imageworks eingesetzt wird, war aber vielleicht noch skeptisch, ob eine frühe Integration sinnvoll wäre. Mittlerweile weiß ich, dass V-Ray auf jeden Fall eine OSL-Implementierung haben wird, das konnte man mir auf der FMX 2013 nochmals bestätigen. Leider konnte man mir noch keine weiteren Details nennen, sprich, man wird noch ein paar Wochen auf eine offizielle Ankündigung warten müssen.

DP: Blender Cycles ist damit immer noch die einzige Pathtracing-Engine mit OSL-Unterstützung. Dafür wurde schon eine große Zahl an Skripten geschrieben. Inwieweit könnten diese Skripte in Zukunft auch eine wertvolle Ressource für Leute werden, die andere Render-Engines nutzen?

Thomas Dinges: Ich denke dass es auf jeden Fall portabel sein wird, das ist ja auch einer der Gründe, warum man eine Shading-Language einsetzen sollte. Man sieht das ja auch an den unterschiedlichen Renderman-kompatiblen Engines, dort kann man die Shader auch gut übertragen. Das einzige, was nicht Teil der OSL-Spezifikation ist, sind die Closures bzw. BRDFs, also die Shader letztendlich. Aber das sind einfache Funktionsaufrufe, die man in den Shadern abändern kann. Wenn man z.B. einen Shader hat, der eine prozedurale Textur implementiert und diese dann mit einem diffusen Shader ausgibt, kann es sein, dass man diesen Aufruf in einer anderen Engine umbenennen muss. Letztendlich sollten die Skripte aber in allen OSL-unterstützenden Render-Engines funktionieren.

DP: Also portabel und in Zukunft sind die im Blender-Umfeld enstandenen Skripte hoffentlich auch für Leute nützlich, die nicht Cycles sondern eine andere Engine nutzen. Du betreibst ja momentan eine Internetpräsenz mit Informationen zur Open Shading Language (www.OpenShading.com), bisher finden sich darauf aber gezwungenermaßen nur Informationen zu OSL in Verbindung mit Blender Cycles. Planst du, die Infos und News auch auf weitere Engines auszuweiten?

Thomas Dinges: Auf jeden Fall. Die Seite OpenShading.com konzentriert sich momentan auf Cycles, da es die einzige Render-Engine ist, die öffentlich zugänglich ist. Ich plane aber, die Informationen auszuweiten, sobald weitere Engines ins Spiel kommen. Ich denke, dass die Seite neben weiteren Ressourcen mit OSL-Skripten einen großen Wert darstellen wird für andere Benutzer, wenn zum Beispiel der OSL-Support in V-Ray eingebaut ist. › ei

Gottfried Hofmann hat an der FAU Erlangen-Nürnberg Informatik studiert. Er arbeitet als Freelancer im VFX-Bereich sowie als Trainer für die freie 3D-Software Blender. Als freischaffender Autor schreibt er für Fach- und Computerzeitschriften. Er hat zahlreiche Blender-Tutorials verfasst, u.a. für CG Tuts+ und CG Cookie. Weiterhin betreibt er die Webseite www.BlenderDiplom.com, auf der fortgeschrittene Blender-Tutorials in deutscher und englischer Sprache zur Verfügung stehen.