Spieleprogrammierung und 3D-Animation Praktikum 4

Allgemeines

Im Praktikum zur Veranstaltung Spieleprogrammierung und 3D-Animation erlernen Sie den praktischen Umgang mit der Spiele-Engine Unity. Es dient als Ergänzung zu den theoretischen Inhalten aus der Vorlesung. Die grundlegenden Konzepte der Spieleentwicklung sind i. d. R. unabhängig von der genutzten Engine, so dass das Wissen, welches Sie bei der Entwicklung mit Unity erlangt haben, leicht auf andere Systeme übertragen werden kann. So sollte Ihnen neben dem Spezialwissen in Unity die Einarbeitung in andere Engines deutlich leichter fallen, wenn Sie das Praktikum erfolgreich absolviert haben.

Zum Bestehen des Praktikums benötigen Sie für das komplette Praktikum mindestens 80% der zu vergebenden regulären Punkte und dürfen kein Aufgabenblatt mit weniger als 50% der Punkte abschließen. Die Punkte pro Aufgabe erhalten Sie nur, wenn Sie beim Testat Ihre Lösungen schlüssig erläutern können.

Bei diesem Aufgabenblatt können Sie regulär 13 und mit Zusatzaufgaben 22 Punkte erreichen.

Das **Testat** zum Aufgabenblatt erfolgt am: **31.05. (Mi) & 02.06. (Fr)**

Thema von Praktikum 4

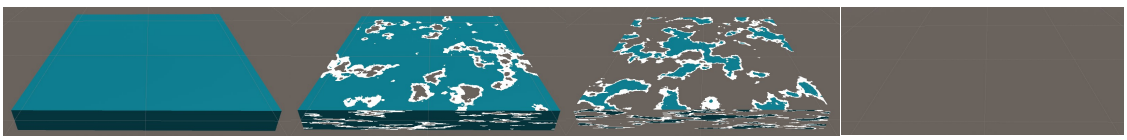
Dieses Aufgabenblatt beinhaltet Übungen zu den Themen Echtzeit-Beleuchtung, Oberflächenmaterialien und Shader-Programmierung. Nach der erfolgreichen Bearbeitung des ABs, wissen Sie, wie Sie mit entsprechenden Materialeigenschaften, Effekten und Beleuchtungseinstellungen die Darstellungsqualität eines Spiels deutlich verbessern können. Sie erlernen insbesondere:

- die dynamische und statische Beleuchtung einer Szene zu parametrisieren,
- Parameter physikalisch basierter Materialien zu bearbeiten,
- eigene Oberflächen-Effekte durch Surface-Shader zu entwickeln,
- Postprocessing-Effekte in Ihr Spiel zu integrieren,
- bestehende Material-Shader in Ihr Projekt zu integrieren und diese zu modifizieren.

Zur Bearbeitung des Praktikums benötigen Sie die Dateien aus dem Archiv Assets_SPA_Praktikum4.zip aus dem ILIAS-Portal oder Team-Raum.

Aufgabe 1 (3 Punkte)

Im ersten Aufgabenblatt sollten Sie das Fall-Guys-Spiel Tip Toe entwickeln, bei dem die Fall-Guy-Spielfigur über Plattformen laufen muss, die dann unter Umständen verschwinden. In dieser Aufgabe sollen Sie einen ansprechenden optischen Effekt für das Verschwinden der Plattformen entwickeln. Statt einfach zu verschwinden, sollen die Plattformen „zerfallen“:

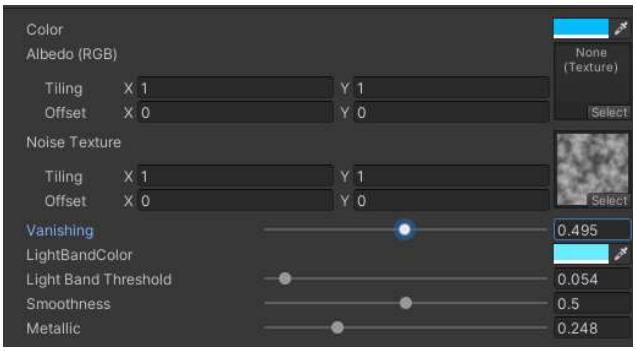


(Die einzelnen Stufen des Zerfallen-Prozesses.)

Die Grundlage für diesen Effekt ist eine Perlin-Noise-Textur, die häufig auch genutzt wird, um das Höhenprofil von künstlichen Berglandschaften nachzubilden (siehe perlinnoise.jpg):

Für den Effekt wird diese Textur durch einen fortlaufenden Grenzwert zwischen [0,1] binarisiert. Die Graustufen der Textur, die kleiner als der Grenzwert sind, sollen dargestellt werden (alpha=1) und die anderen Werte sollen nicht dargestellt werden (alpha=0). Wandert der Grenzwert also von 1 zu 0, löst sich das Objekt auf.

Der Shader soll über die folgenden Parameter steuerbar sein:



Das Material unterstützt die Einstellung einer Farbe (*Color*) und einer Albedo-Textur, zusätzlich sollen die Parameter *Smoothness* und *Metallic* aus dem Standard-Surface-Shader von Unity übernommen werden. Darüber hinaus soll die Noise-Textur (hier perlinnoise.jpg) übergeben werden können. Der Parameter *Vanishing* entspricht dem oben beschriebenen Grenzwert zur Binarisierung der Noise-Textur. Die Parameter *LightBandColor* und *LightBandThreshold* steuern den Übergangsbereich zwischen „sichtbar“ und „unsichtbar“ (der weiße Bereich in der obigen Bildfolge). *LightBandThreshold* gibt in einem Bereich von [0,1] an, wie „breit“ und *LightBandColor* in welcher Farbe der Übergangsbereich erscheinen soll. Überlegen Sie, wie Sie diesen Übergangsbereich programmieren können. Die *LightBandColor* soll auf den Emission-Kanal des Materials gesetzt werden.

Erstellen Sie einen Surface-Shader, der den entsprechenden Effekt implementiert.

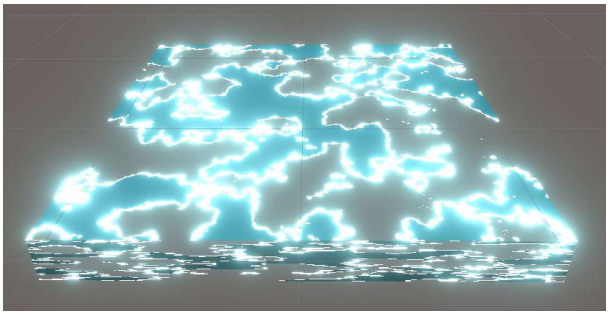
Tipp: Der Surface-Shader verwendet den Alpha-Kanal des Materials, um dieses „durchsichtig“ zu machen. Hierfür müssen Sie die folgenden Tags im Shader definieren:

```
Tags { "RenderType"="Transparent" "IgnoreProjector"="True" "Queue"="Transparent" }
```

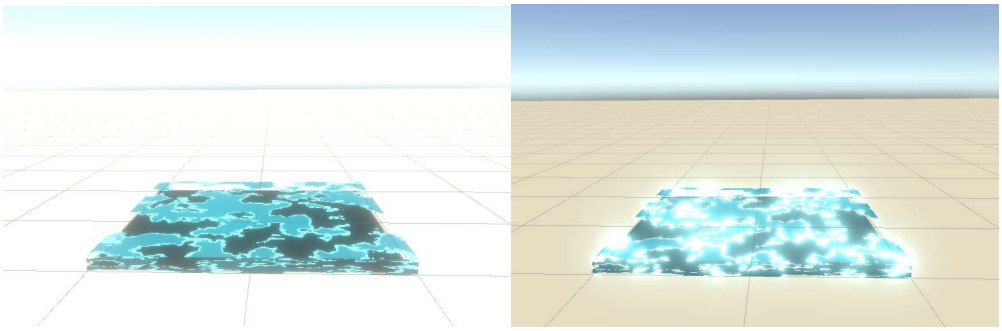
Darüber hinaus müssen Sie die #pragma-Anweisung im Shader um `alpha:fade` ergänzen.

Aufgabe 2 (2 Punkte)

Erweitern Sie Ihre Szene um einen Bloom-Postprocessing-Effekt, der das LightBand aus Aufgabe 1 (Übergangsbereich im Emission-Kanal) glühen lässt:



Sorgen Sie durch entsprechende Parametrisierung des Bloom-Effekts und durch Verwendung von HDR-Farben (Farbwerte >1) im Emission-Kanal des „Auflösen“-Effekts dafür, dass nicht die ganze Szene „überbeleuchtet“ wird, sondern nur der entsprechende Bereich des Materials:



(Bild links: Falsch eingestellter Bloom-Effekt. Die Szene wird als Ganzes „überbeleuchtet“. Bild rechts: Der Bloom-Effekt wird nur auf den Übergangsbereich des Effekts angewandt, da dieser deutlich hellere Emissions-Farbwerte hat, als der Rest der Szene.)

Um Postprocessing-Effekte in Ihrer Szene nutzen zu können, müssen Sie im Window->Packet Manager das „Post Processing“ Package installieren, eine Post-process Layer Component zur aktiven Kamera hinzufügen, einem Objekt in der Szene eine Postprocessing Volume Component hinzufügen und diesem wiederum ein Post-process Volume zuweisen, welches dann den Bloom-Effekt enthalten muss.

Tipp: Spielen Sie mit den Parametern *Intensity* und *Theshold* des Bloom-Effekts.

Extra Achievement 1 (2 Punkte)

Binden Sie das neue Material in die Skript-Logik der Plattformen aus Praktikum 1 ein.

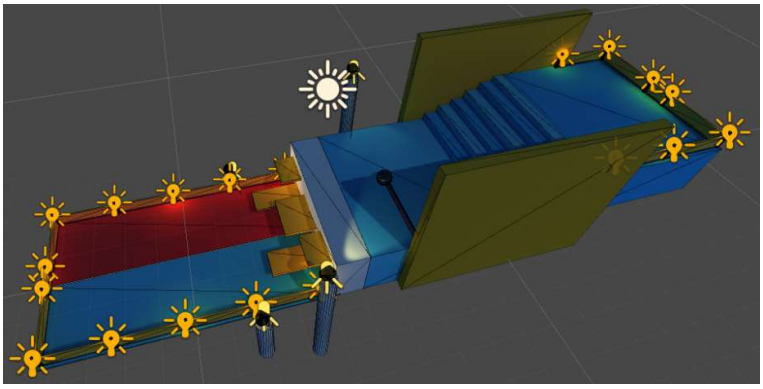
Aufgabe 3 (2 Punkte)

Importieren Sie das P4_Lighting Package aus dem Verzeichnis. Die dort enthaltene *Lighting_Clean* Szene enthält ein sehr einfaches Level. Diese Szene sollen Sie nun mit einer Mischung aus dynamischer und statischer Beleuchtung versehen.

Die Szene enthält mehrere Objekte die Ihnen als Orientierung dafür dienen sollen, wo welche Lichtquelle positioniert werden muss. *pLightObjects* enthält alle Objekte an deren Stelle Punktlichter sitzen sollen, *sLightObjects* zeigt dagegen wo und in welche Richtung Spotlights sitzen sollen.

Erstellen Sie die Lichtquellen und geben Sie ihnen einen angemessenen *Radius*. Für die *Lichtfarbe* orientieren Sie sich an den Texturen der Objekte. Stellen Sie sicher, dass das Mesh der modellierten Lichter nicht das Licht Ihrer Lichtquellen aufhält. Die Spotlichtquellen sollten Sie so ausrichten, dass sie in etwa zum Winkel der Scheinwerfer passen. Stellen Sie auch die *Intensity* passend ein.

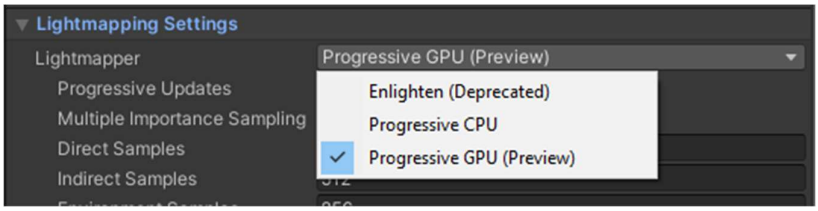
Ihre Lichter sollten also grob so verteilt sein, wie auf folgendem Bild zu sehen:



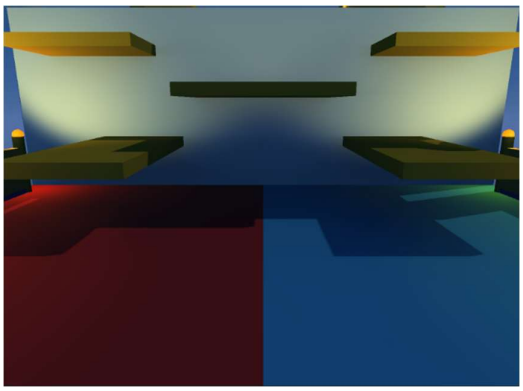
Ein großer Teil der Szene besteht aus unbeweglichen Objekten. Für diese Objekte kann ein Teil der Beleuchtung im Voraus generiert werden. In diesem Fall soll das nur für das direktionale Licht und die Spot Lichtquellen passieren.

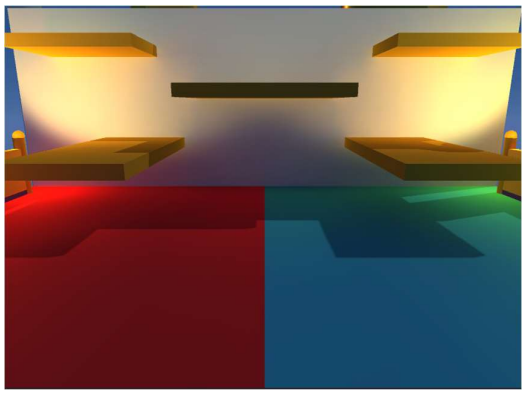
Dafür müssen Sie zum einen alle unbeweglichen Objekte auf *static* setzen. Wählen Sie nun die Lichtquellen in der Szene aus und stellen Sie den *Mode* auf *Mixed*. Es kann sein, dass Unity nun automatisch beginnt das Licht zu backen. Sollte es das nicht tun, können Sie über Window->rendering->Lighting die Lighting Settings öffnen. Dort können Sie unter Scene dann entweder manuell *Generate Lighting* drücken oder den Haken bei *Auto Generate* setzen.

Tipp: Wenn Sie eine gute Grafikkarte haben, können Sie in den Lightmapping Settings den Lightmapper auf Progressive GPU stellen, dann geht die Berechnung sehr viel schneller:



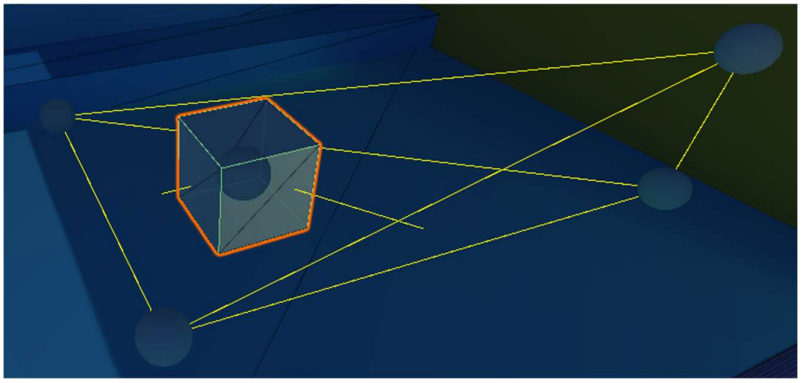
Nachdem der Prozess fertig ist (das kann durchaus 10 Minuten oder mehr dauern, falls Sie zum Beispiel an einem älteren Laptop arbeiten), sollte Ihre Game View sehr anders als zuvor aussehen. Besonders an der weißen Wand sollte rechts ein blauer, und links ein roter Farbton zu erkennen sein. Hier ein Beispiel wie es vor- und nach dem Backen aussehen könnte:





Aufgabe 4 (2 Punkte)

Bisher haben Sie die statischen Objekte beleuchtet, doch Ihr Charakter und die beweglichen Objekte sollen von diesem vorgenerierten indirekten Licht ebenfalls profitieren. Dies können Sie durch das Hinzufügen von *Light Probes* realisieren. Achten Sie bei der Verteilung der *Light Probes* darauf, dass sich in den Lichtübergängen (zum Beispiel Farbwechsel oder Licht zu Schatten) mehrere *Probes* befinden.



Nachdem Sie die *Probes* gesetzt haben müssen Sie das Licht erneut generieren.

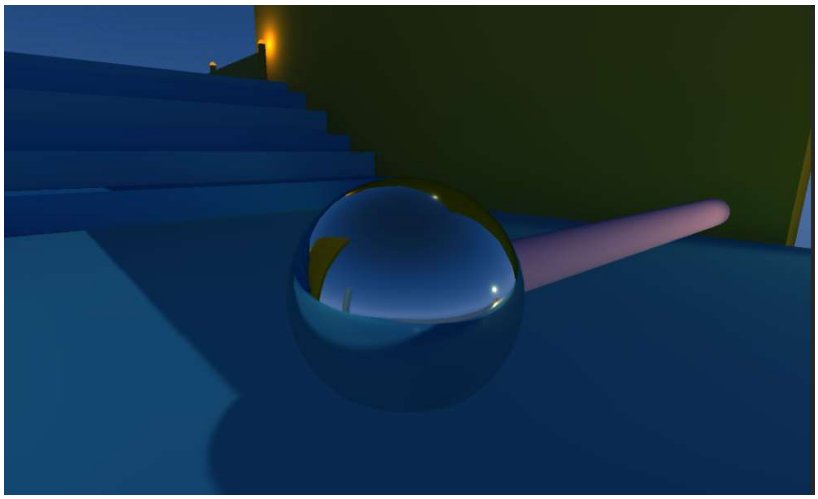
Wenn sie danach ein Objekt in die Szene setzen und hin und her bewegen, sollte Ihnen angezeigt werden, welche light *Probes* für die Beleuchtung verwendet werden.

Tipp: Die Probes dürfen nicht in anderen Objekten stecken.

Tipp: Falls Ihr Licht nach dem Auto generieren komisch aussieht, kann ein löschen und neu generieren des Baked Lighting helfen.

Extra Achievement 2 (1 Punkt)

In dem Level gibt es ein Hindernis auf der oberen Ebene. Das Zentrum davon ist eine metallene Kugel. Passen Sie das Material dieser Kugel so an, dass es ein sehr glattes, metallisches Material wird. Legen Sie sich für die beiden Objekte eine passend große *Reflection Probe* an. Ihr Ergebnis könnte in etwa so aussehen:



Aufgabe 5 (2 Punkte)

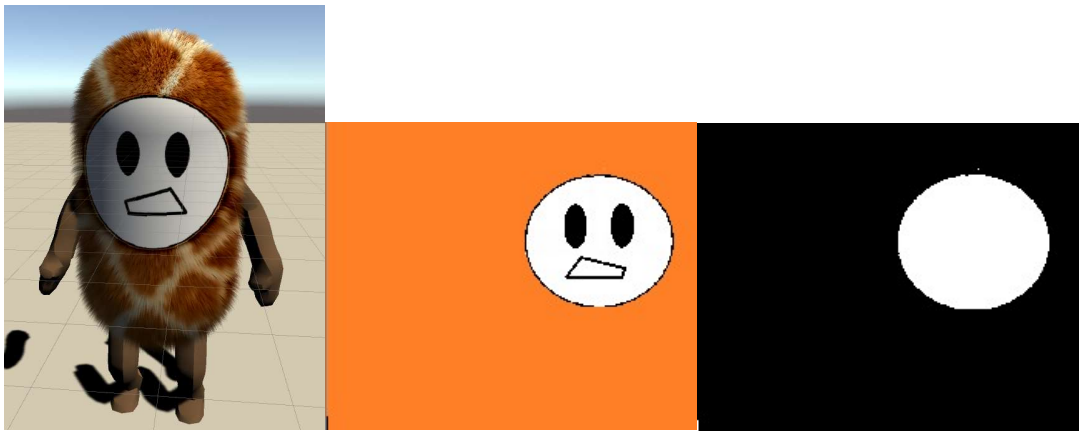
Ihre Fall-Guy-Figur soll ein Fell bekommen, hierfür wurde ein entsprechender Shader aus dem Internet heruntergeladen (FurShader.shader & FurHelper.cginc aus SP3DA_Dateien_AB4.zip). Integrieren Sie den Shader, parametrisieren Sie ein korrespondierendes Material in Ihre Szene und wenden Sie es auf den Körper Ihrer Fall-Guy-Figur an:

(Spielen Sie mit den Parametern des Shaders. Die entsprechenden Texturen finden Sie ebenfalls in SP3DA_Dateien_AB4.zip.)

Betrachten Sie die Funktionsweise des Shaders. Sie müssen die Funktionsweise des Effekts erläutern können. Welche Eigenschaften des Effekts sehen Sie kritisch und warum?

Aufgabe 6 (2 Punkte)

Erweitern Sie den Fell-Shader aus Aufgabe 3 so, dass eine weitere Farb-Textur mit Alpha-Kanal verwendet werden kann. Der Alpha-Kanal soll steuern, wo Fell erscheint (alpha=1) und wo nicht (alpha=0). An Stellen, an denen kein Fell ist, soll die Farbe aus der neuen Textur erscheinen:



(links: Ergebnis, Mitte: RGB-Kanäle der zusätzlichen Textur, rechts: Alpha-Kanal der zusätzlichen Textur).

Erstellen Sie für Ihren Fall-Guy eine lustige Frisur, indem Sie mit den Texturen und dem Material spielen.

Tipp: Um einen Alpha-Kanal in einer Textur abspeichern zu können, muss das Format der Textur einen solchen unterstützen. Dies ist zum Beispiel beim TGA-Format der Fall.

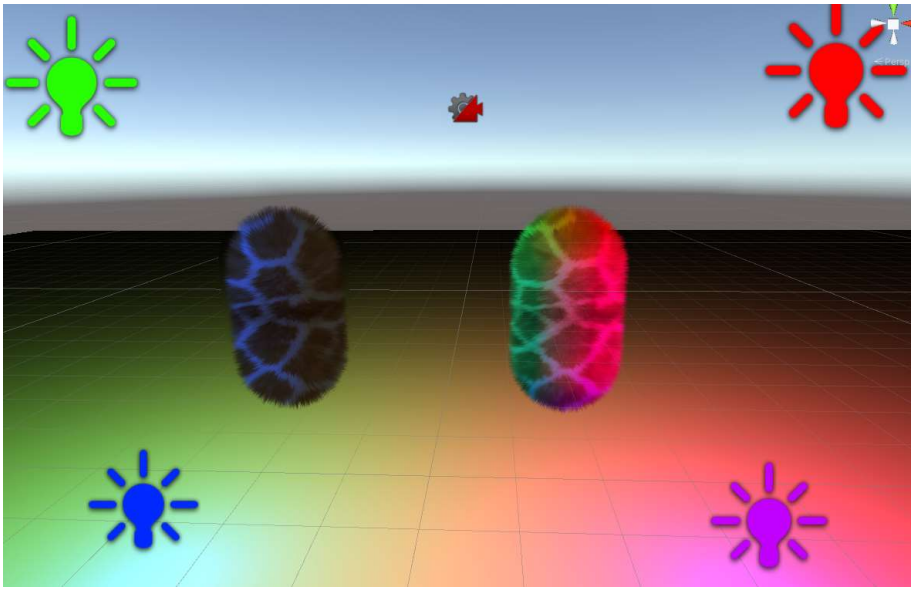
Extra Achievement 3 (2 Punkte)

Der Fell-Shader unterstützt über den Parameter „Global Force“ die Einbringung einer Kraft auf das Fell. Schreiben Sie ein Skript, welches das Fell physikalisch nachbewegt, wenn sich der Körper des Fall-Guys bewegt:

(links: stehend, die Haare werden durch die Schwerkraft nach unten gezogen, Mitte: nach vorne rennend, die Haare werden durch ihre Trägheit und den Windwiderstand nach hinten gezogen, rechts: fallend, die Haare werden durch ihre Trägheit und den Windwiderstand nach oben gezogen.)

Extra Achievement 4 (4 Punkte)

Leider unterstützt der Fell-Shader aus dem Internet nicht das vollständige Beleuchtungsmodell von Unity, da das Standard-Beleuchtungsverhalten von diesem nicht berücksichtigt wird. So unterstützt der Shader nicht mehrere Lichtquellen und kein korrektes HDR-Verhalten:



(Rechts: Fell-Shader aus dem Internet, der nur eine Lichtquelle unterstützt. Links: angepasster Fell-Shader, der korrekt mit der physikalisch basierten Standard-Beleuchtung aus Unity funktioniert.)

Schreiben Sie den Shader so um, dass er mit dem Standard-Beleuchtungsmodell von Unity korrekt funktioniert.

Tipp: Surface-Shader erlauben die Nutzung des Standard-Beleuchtungsmodells von Unity.

Tipp: Sie können in Surface-Shader nicht mehrere „Pass“-Blöcke verwenden, aber Sie können auch ohne Pass-Block mehrere Shader-Programme (CGPROGAM..CGEND CGPROGRAM..CGEND etc.) in einer Shaderdatei hineinander ausführen.