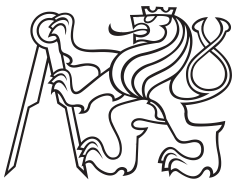
Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Materials in Computer Graphics

Material Converter

Tomáš Cicvárek

Supervisor: Ing. David Sedláček, Ph.D.
Field of study: Open Informatics
Subfield: Computer Games and Graphics
May 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Cicvárek** Jméno: **Tomáš** Osobní číslo: **466021**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Materiály v počítačové grafice

Název bakalářské práce anglicky:

Pokyny pro vypracování:

Seznamte se s běžně používanými modely pro popis materiálových vlastností objektů (např. empirické, PBR a jejich konkrétními variantami jako Phong) s ohledem na využití v realtime grafickém zobrazování herního enginu Unity a fotorealistického rendereru Octane firmy Otoy, který slouží jako offline renderer (raytracer) pro Unity.
Popište formáty používané pro zápis těchto materiálů, přičemž se soustřeďte na formáty používáne při tvorbě her pro přenos modelů (např. obj, fbx, gltf). Problémem využití Octane rendereru s Unity je nevhodná automatická konverze materiálů z formátů, které unity importuje do vnitřního unity formátu s následnou konverzí do formátu Octane. Tato konverze často vede na nutnou ruční editaci Octane verze daného materiálu (oprava automatické konverze).
Analyzujte možnosti existujících konverzních postupů a unifikovaných popisů materiálů. Navrhněte a implementujte konverzní nástroje pro vybrané třídy materiálů z původního modelu do Octane popisu materiálu, tak aby se minimalizovali časté ruční zásahy do konverzního postupu.
Konverzní skripty implementujte v podobě distribuovatelného balíčku (package). Funkcionalitu ověřte na široké škále vstupních materiálů.

Seznam doporučené literatury:

1] J. Žára, B. Beneš, J. Sochor, P. Felkel, Moderní počítačová grafika, 2. vydání. Computer Press, 2004.
2] M. Pharr, W. Jakob, G. Humphreys, Physically Based Rendering: From Theory to Implementation, 3rd Edition. Morgan Kaufmann, 2016. dostupné online: http://www.pbr-book.org/
3] Wes McDermott, The PBR Guide, 2018. dostupné online: https://academy.substance3d.com/courses/pbrguide
4] B. Burley, Physically-Based Shading at Disney, SIGGRAPH 2012. dostupné online: https://www.disneyanimation.com/publications/physically-based-shading-at-disney/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Sedláček, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **18.03.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to give my thanks to my supervisor, Ing. David Sedláček, Ph.D., for his supervision over my thesis and for all information and consultations provided by him. I would like to thank my family as well for their psychical support.

Declaration

I hereby declare that the present bachelor's thesis was composed by myself only and that I specified all used resources in accordance with the Methodical guideline for adhering to ethical principles when I was working on the academic final thesis.

Prague, 20. May 2021

Abstract

The purpose of this thesis is to describe formats used in real-time graphic view of game engine Unity and photo-realistic render Octane created by the company Otoy. Octane is used by Unity as an offline render (ray-tracer). Furthermore, conversion methods are analyzed and implemented in scripts used by Unity.

Keywords: materials, material conversion, Unity, C#, Octane

Supervisor: Ing. David Sedláček, Ph.D.

Abstrakt

Téma této práce zahrnuje popis formátů používaných v real-time grafickém zobrazování herního enginu Unity a fotorealistického vykreslovacího softwaru Octane firmy Otoy, který slouží jako offline vykreslovací software (ray-tracer) pro Unity. Dále jsou analyzovány konverzní postupy a jsou implementovány ve skriptech pro Unity.

Klíčová slova: materiály, konverze materiálů, Unity, C#, Octane

Překlad názvu: Materiály v Počítačové Grafice — Konvertor Materiálů

Contents

Chapter 1

Introduction

In computer graphics, we use materials to describe how the surface of models interacts with light. Simply put, materials describe the appearance of the models. For that purpose, many formats are made. One of such formats is brought to the Unity engine by the Octane render.

Octane is an unbiased spectrally correct GPU render engine, which allows modification of the scene in real-time. Octane is gaining popularity among 3D artists and in the film industry for its capabilities and performance. Although Octane is not fast enough to be used in games, it is fast enough to quickly make pre-rendered scenes and movies with it and modify those in real-time while rendering.

To use Octane in Unity, Octane for Unity plugin is needed. This plugin adds a new environment to Unity that can satisfy most scene developers and enables them to make high-quality scenes with physically-believable 3D models and materials.

While Unity supports many 3D file formats that are converted to the format used by Unity, due to the fact that Octane is provided from an external source, there is no conversion tool available in Unity for the materials provided by Octane. Because of that, those materials must be created manually anew one after another.

The absence of a conversion tool between the standardized materials and the Octane materials lead to the purpose of this thesis, which is to analyze

the materials provided by the Octane for Unity plugin, their capabilities and description, and to devise and implement methods to make the automatic conversion from standardized materials to Octane materials possible in Unity.

Chapter 2

Materials in Computer Graphics

In computer graphics, we use materials to describe the look of surfaces of three-dimensional objects. Materials describe how light interacts with the surface of an object. From the categories of optics, science about light, we use are primarily geometrical optics and sometimes we describe light as particles [JrF05]. Geometrical optics describe light as beams that traverse through space and can be described with geometric rules.

2.1 Empirical illumination Models

Empirical illumination models are models that use empirical observations for the computation of the lighting in three-dimensional scenes. Their advantage lies in their computation complexity that is lower than in cases of physically-based illumination models. Empirical models can be described as reflection models that *"account for masking and self-shadowing effects and predict off-specular reflection."* [KE09]

I present here three examples of the empirical models.

2.1.1 Lambertian reflection model

Lambertian reflectance does not depend on the viewing direction. Because of this fact, this reflection model is unable to compute specular highlights and thus allows us to create diffuse and ambient lights only. Lambertian reflection in one point is computed by using the normal vector in the point on the surface, the viewing ray and the vector representing an incident ray from the light source, the ray from the light source would pass through the surface. The reflection occurs if the angle between the viewing ray and the normal vector is equal to the angle between the normal vector and the incident ray. For further detailed information visit [Kop14].

2.1.2 Phong reflection model

This reflection was designed by Bui-Tuong Phong in 1977. This reflection model distinguishes between three types of light (Figure 2.2) [JrF05]:

- Ambient light
- Diffuse light
- Specular light

The ambient light is a light that does not come from any light source. It can be viewed as a light reflected so many times that the direction from which the light has come is unknown [Eck18]. The light is thus equally spread all over the surface of the illuminated object. Let I_a be the intensity of ambient light, k_a its coefficient and L_a the ambient light color. The intensity of ambient light is computed as

$$I_a = k_a L_a$$

where the intensity of the ambient light remains generally constant for the whole scene.

The diffuse light on the other hand depends on the direction of its light source. The assumption is that this light is reflected, the reflection is equally spread in all possible directions. Let I_d be the intensity of the diffuse light,

k_d its coefficient, L_d materials diffuse color, $\vec{L}$ is a normalized vector from a light source and $\vec{N}$ is a normalized normal vector on a surface. If $(\vec{N} \cdot \vec{L}) < 0$, then the point on the surface cannot be illuminated by the light source. Then the resulting diffuse light intensity on the surface will be computed as [JrF05]

$$I_d = k_d \max(0, \vec{N} \cdot \vec{L}) L_d$$

which equals the Lambertian reflection. The specular light depends on the

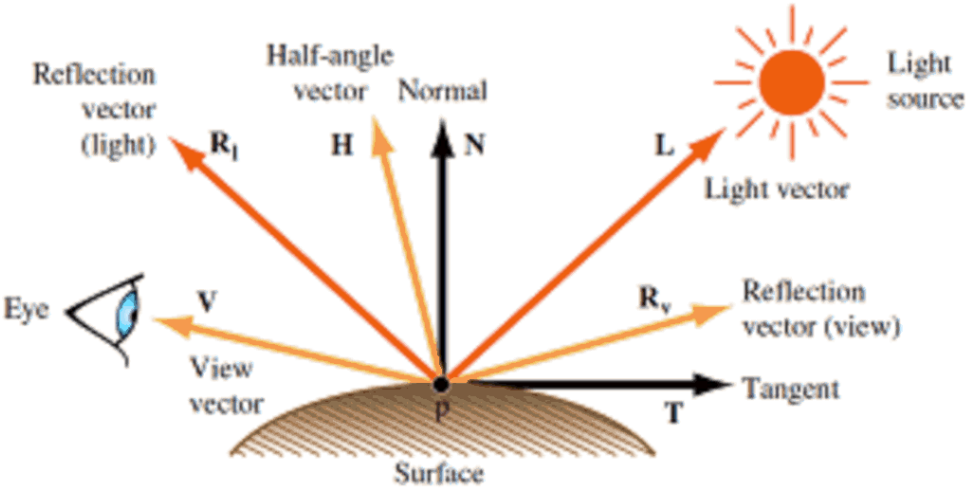


Figure 2.1: Ray vectors used in reflection model [Kim].

reflection vector and the view (eye) vector. This light component is responsible for highlights that can be seen in the scene. Let I_s be the specular intensity, k_s the specular coefficient, s the specular exponent, L_s the specular color, $\vec{R}$ the normalized reflected ray, and $\vec{V}$ the normalized view ray. The specular intensity will be computed as [IPI13]:

$$I_s = k_s (\vec{R} \cdot \vec{V})^s L_s$$

The viewed resulting light is composed of those three light components [JrF05]

$$I_v = I_a + I_d + I_s$$

where I_v is the viewed light.

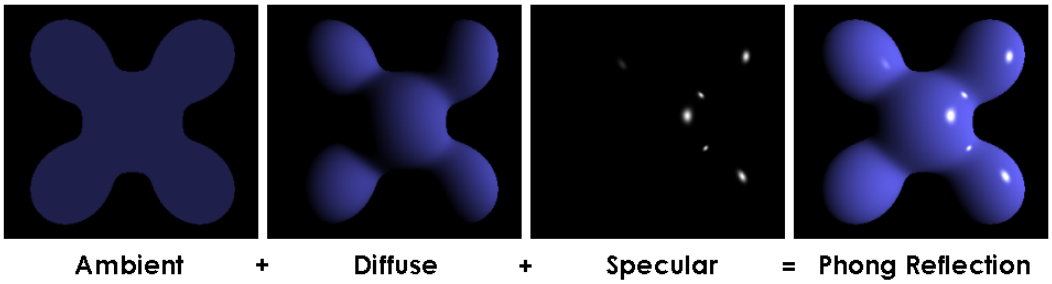


Figure 2.2: Composition of Phong reflection model [Smi].

2.1.3 Blinn-Phong reflection model

Blinn-Phong reflection model is a modification of the Phong reflection model made by Jim Blinn (Figure 2.3). According to [Bli77], this model solves issues with reflections from the Phong reflection model by using the halfway vector while computing specular light. This vector is computed as

$$\vec{H} = \frac{\vec{L} + \vec{E}}{||\vec{L} + \vec{E}||}$$

where $\vec{L}$ is the direction vector to the light, $\vec{H}$ is the halfway vector and $\vec{E}$ is the vector of direction to the eye (the synonym to the eye vector is a view vector $\vec{V}$), which can be seen in Figure 2.1. The resulting Phong equation changes as such:

$$I_v = I_a + I_d + S * k_s L_s$$

where $S = (\vec{N} \cdot \vec{H})^s$, s is the specular exponent.

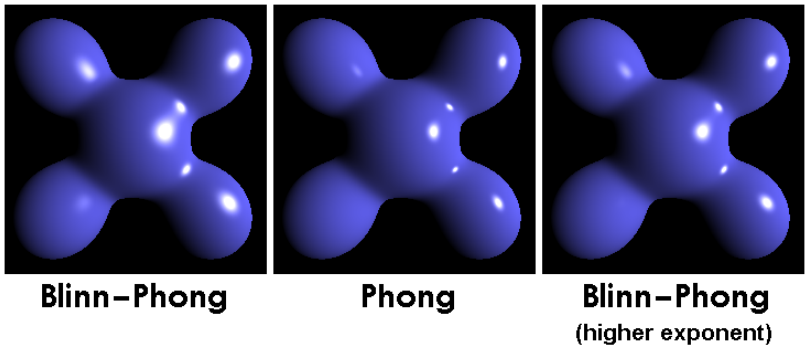


Figure 2.3: Comparison between Blinn-Phong and Phong reflection models [Rai].

2.2 PBR (Physically Based Rendering) models

PBR models tend to make their materials Physically-based, meaning that they are physically-plausible. To achieve believable results, we will need a few terms [TJL, JrF05]:

- Radiance
- BRDF
- Microfacets

Radiance is a unit used for global lighting. This unit indicates how much light was received and emitted per unit solid angle per unit perpendicularly projected area. We can simplify this unit as a ray color. The radiance is constant in a vacuum and this fact is used in global lighting algorithms excluding participating media like smoke, fog, etc. [JrF05] BRDF (Bidirectional Reflectance Distribution Function) describes the reflectivity of a material. BRDF "returns the percentage of how much of the incoming light in a given direction is reflected towards another given direction at a surface point." [TJL] BRDF has those assumptions [JrF05]:

- The light that is going to be reflected is reflected immediately.
- A photon of a specific wavelength is reflected with the same wavelength.
- The light ray that is directed to a certain point on the surface is reflected from the same point. If not, that would mean that the light traveled under the surface. This behavior is described with BSSRDF (Bidirectional Scattering Surface Reflectance Distribution Function), which is omitted in BRDF.

The following text is based on [JrF05].

BRDF is always positive and is anisotropic. The second term means that the reflected light does not depend only on the direction of the light that is received or reflected, but on the surface rotation around the normal vector of this surface as well. BRDF adheres to the energy conservation law. Thus the amount of light that hits the surface is either reflected, transmitted or absorbed.

PBR assumes the existence of the microfacets, which are small surfaces that cast shadows over each other (in cases of non-reflective or partly reflective materials) generally in the shape of V-cavities. The V-cavities might not be necessarily symmetrical in both directions (as in Figure 2.4), but in one direction only. The whole surface is expected to be made out of them. Microfacets thus have the ability to describe how reflective the surface is.

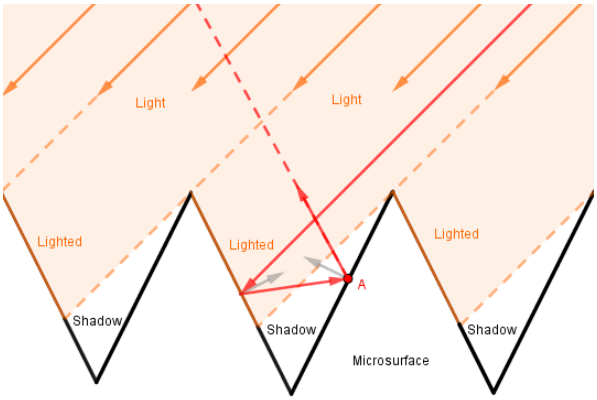


Figure 2.4: Microfacets cast a shadow over each other and describe the reflectivity of the material. [TJL]

2.2.1 Torrance-Sparrow reflection model

This section is based on the text from [PJH16, MU12]. Torrance-Sparrow model was developed in 1967 to describe isotropic (e.g metallic) materials. This PBR model uses the microfacet theory. Microfacets are distributed on the surface according to set parameterization and their orientation is random. As well as in the Blinn-Phong reflection model, the half-angle (previously mentioned as halfway) vector is used instead of the surface normal, thus the deviation between the microfacet normal vector and the half-angle vector is measured instead.

According to the [MU12], the BRDF is computed as follows:

$$f_r(\vec{L},\vec{V}) = \frac{k_d}{\pi} + \frac{k_s}{4\pi(\vec{N}\cdot\vec{V})} D(\vec{H})F(\vec{L})G(\vec{L},\vec{V})$$

where k_d and k_s are diffuse and specular coefficients, $\vec{L}$ is a ray from the source, $\vec{H}$ is the half-angle vector and $\vec{V}$ is the view vector. All the vectors can be seen in Figure 2.1. The functions $D(\vec{H})$, $F(\vec{L})$ and $G(\vec{L},\vec{V})$ are described as:

- $D(\vec{H})$ - Gives distribution of the normals of the microfacets aligned with the half-angle vector. The functions primarily used are the Beckmann or the Gaussian distribution functions.
- $F(\vec{L})$ - "Fresnel factor gives the fraction of light that is reflected from the entire surface." [MU12]
- $G(\vec{L},\vec{V})$ - Geometric attenuation factor describes the shadowing and masking on the surface.

This model has put a base for many models based on the microfacet theory.

2.2.2 Cook-Torrance reflection model

R.L.Cook and K.E.Torrence put together a specular reflectance BRDF model based on the Torrence-Sparrow model. The improvement this model has is that only the microfacets oriented towards the half-angle vector contribute to the BRDF. Moreover, this model distinguishes between the metallic and the non-metallic surfaces and optimizes the computation process of the Fresnel function.

There are some limits to this BRDF. The most important one is that this model does not adhere to the energy conservation law at certain angles.

For more information visiting [TJL, MU12] is recommended.

Chapter 3

Material formats

If we want to store data about 3D models, we save those data to files of certain formats. Many of the standardized 3D formats make it possible to share our models between other applications and platforms. A lot of those formats can save the whole scene: lights, geometry, materials, animations, cameras, etc. On the other hand, there are formats that enable saving only some of the scene parts that were mentioned previously.

There are two ways how to save the data. Several formats save data as ASCII characters, that are readable in text editors. The other way is to save the data in a binary file, which saves space, but specialized software is needed in order to read them. Some formats use both ways of saving 3D data.

Establishing standardized formats put down the common ground for many 3D applications of how to save material properties. Each of them can, however, save the material properties in their own way or skip them. Such behavior can be seen in 3D tools based on PBR rendering and formats that enable empirical reflection models only. In such cases, some properties are missing in the format or there are properties that remain unused.

3.1 Wavefront Format

This format contains the definition of the 3D models only, namely the geometry, the UVs and the materials, and saves them as ASCII characters in two types of files. The geometry and the UVs are saved separately in an *.obj*

file with references to the materials, which are all saved in a file with the extension *.mtl*.

By following the documentation [DR95], the material is described by using a list of statements with operators, which are voluntary. Those statements contain:

1. Name of the material which follows the statement `newmtl`. The `newmtl` statement separates the materials as well.
2. Material color and illumination statements contain the color, transparency and reflectivity values. The most used ones are:
   - `Ka` - ambient color (basic are RGB values in the range from 0 to 1)
   - `Kd` - diffuse color (basic are RGB values in the range from 0 to 1)
   - `Ks` - specular color (basic are RGB values in the range from 0 to 1)
   - `illum` - enum value deciding, which illumination model should be used (there are values available in the range from 0 to 10)
   - `d` - opacity of the material (the value is in the range from 0 to 1, where 1 describes fully opaque material while 0 fully transparent one)
   - `Ns` - specular exponent with the value from 0 to 1000
3. Texture map statements are accompanied by a file path to an image texture. The most commonly used ones are:
   - `map_Ka` - ambient texture
   - `map_Kd` - diffuse texture
   - `map_Ks` - specular texture
   - `map_Ns` - map connected to the specular exponent
   - `map_d` - opacity map
   - `bump` - bump map
4. The reflection map is an environment that can be in the shape of a cube or a sphere. This environment specifies reflections on the material and is described by a texture. It can be imagined as a skybox used only by the specified material.

The materials are saved according to the empirical reflection models. This leaves 3D editing tools based on PBR reflection models in a situation, where there are unused and missing statements. However, since the foundation of this file format there have been attempts, and some of them were successful, in establishing new statements according to the PBR statements as well. According to the [WMT20] there were added these statements:

- `Ke` - emission color
- `map_Ke` - texture describing emission color
- `norm` - normal map

The other statements in the list described at [Hou15] were not officially added.

3.2 FBX Format

FBX is one of the most famous formats and is one of the formats that are supported by Unity [Unib]. According to [Hou], this format was created in order to share high-quality 3D data between different tools.

There is an SDK available to the public, however, the license to the FBX itself is fully closed [TON13]. More about FBX can be found on the Autodesk documentation website [Aut]. The FBX has both binary and ASCII versions and can contain the whole scene including cameras and lights. The position, UV coordinates and normals that have different topology are enabled in this format as well. The downside to this format is that FBX format uses a historical lighting model, materials are supposed to be saved in an empirical reflection model Blinn-Phong, and enables only one texture per material property [Hou]. By observing the ASCII formatted FBX file was deducted that the FBX document is a nested list of nodes [TON13].

3.3 Octane PBR Override

There are many material shaders available in Unity. With the Octane for Unity plugin developed by the company Otoy Inc., there is added another one, the Octane PBR Override shader.

Octane is an unbiased physically-based GPU render engine [Bri10]. Its work with GPU makes the rendering faster than other render engines that use CPU for rendering computation.

Biased renders use methods to optimize the computation of the rendered image, such as limiting the number of reflections. On the other hand, the unbiased renders do not use those "shortcuts". An example of such behavior

is that the unbiased renders wait until the light rays are fully absorbed or until they leave the scene entirely. Although those methods allow creating high-quality materials, those materials might not be necessarily physically correct and their computation time much higher than in the case of the biased renders. The term used is that those materials are physically believable means that the material is not always described by accurate values, but its appearance is close to reality and the difference from real materials is almost unrecognizable [Nic16].

3.3.1 Loading Octane in Unity

Octane for Unity renders the scene in a separate tab which is called PBR Viewport. To start rendering in this tab, a few steps must be followed:

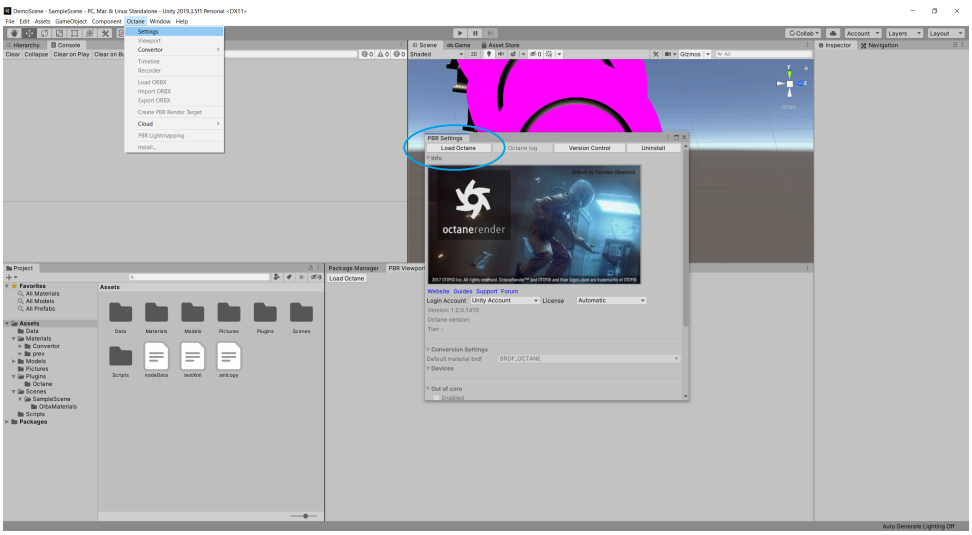


Figure 3.1: The first step is to load Octane render.

1. Load the plugin (Figure 3.1) - After adding the Octane for Unity to a project, a new item will appear in the top panel. This item, Octane, contains an item Settings. After clicking on this item, a window appears. There can be seen a button with the label Load Octane which starts the Octane functionality.

2. Create a render target (Figure 3.2) - Press the right mouse button in the hierarchy. Find the Create PBR Render Target item and select it.

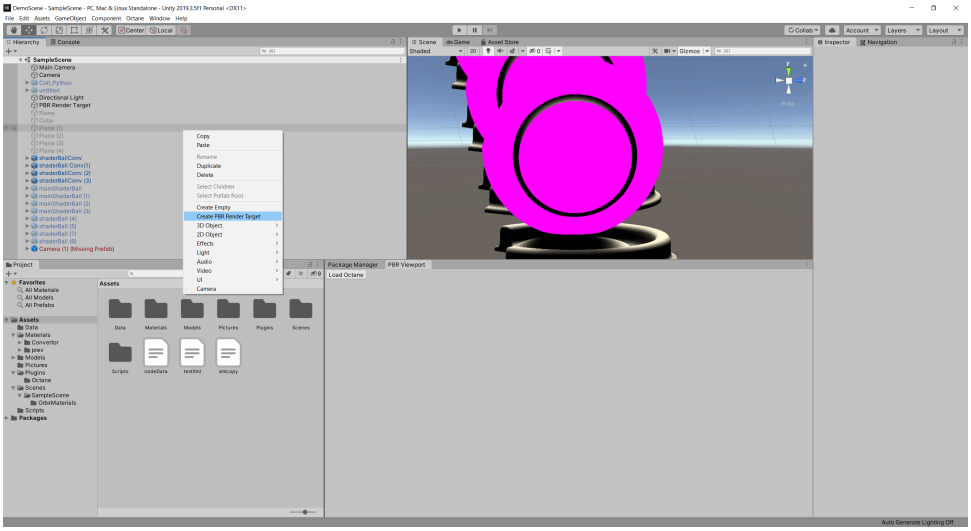


Figure 3.2: Create a render target.

3. Begin the rendering process (Figure 3.3) - Now that the Octane render is loaded and there is a render target available, the rendering process can begin. Selecting the PBR Render Target in the scene hierarchy shows properties of the PBR Render Target in the Inspector tab. There is a button labeled as Render. After that button is clicked on, the rendered scene appears in the PBR Viewport tab. This tab appears itself if it was not opened before or if it was closed before the rendering process started.

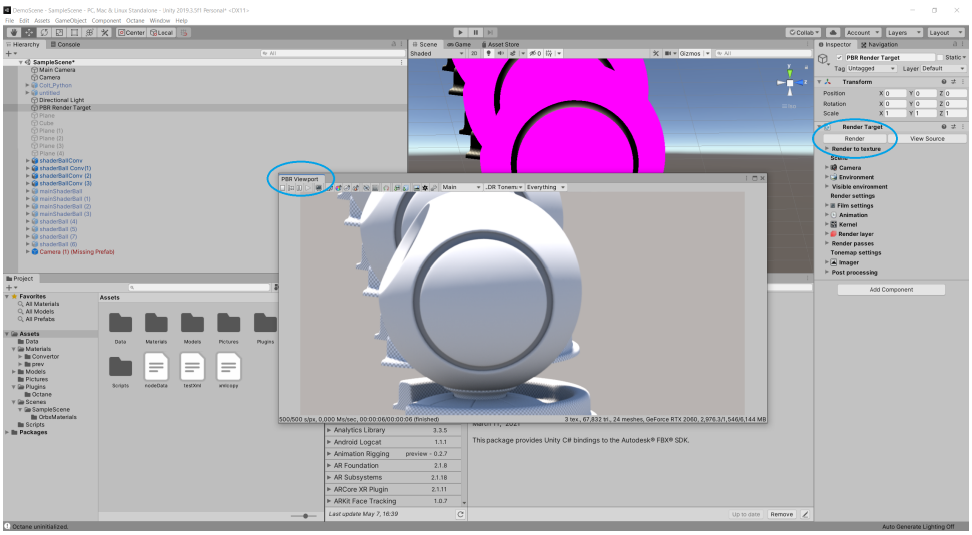


Figure 3.3: Start the rendering process. The closed PBR Viewport tab is opened automatically.

3.3.2 PBR Override material

The PBR Viewport tab is the only scene window in Unity capable of rendering the Octane materials. The Scene tab or the Game tab cannot render these materials. The PBR Viewport is compatible with two material shaders native to Unity, the Standard and the Standard (Specular setup) shaders [Oto].

The materials can be examined and modified by using a source window that can be opened in the Inspector tab after selecting the desired material (Figure 3.4).

The whole material is saved in a material file and saved as an Octane XML. The XML consists of the following five types of nodes where only the top three node types have influence over the appearance of the material. Their description can be compared to the picture above (Figure 3.4):

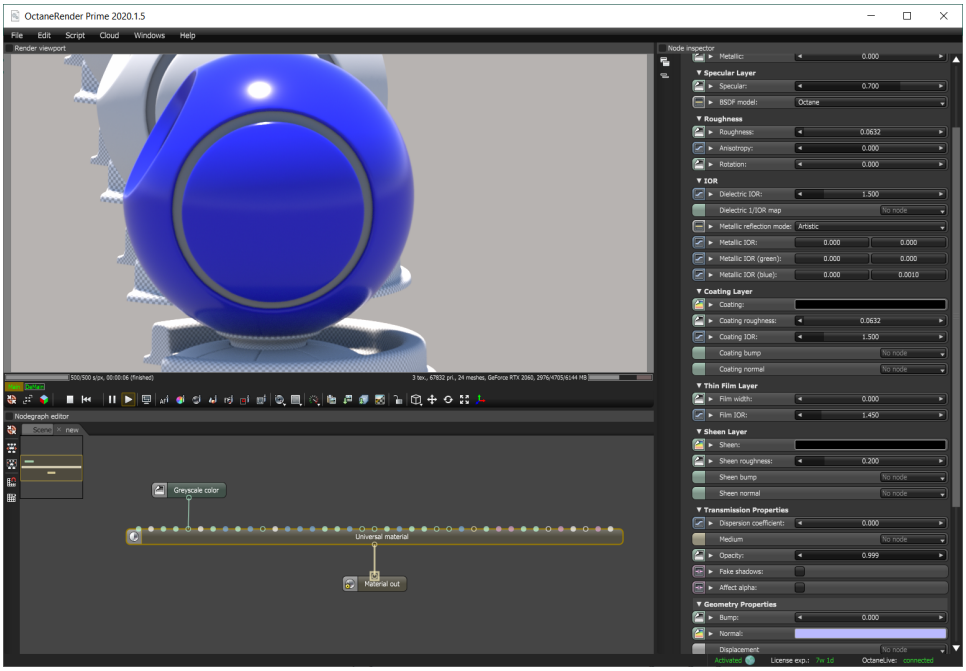


Figure 3.4: The window used for managing an Octane material is divided into three parts. In the left bottom part, there is a working area where is the graph describing the material. In the right part, there is an area used for inspecting the selected node in the graph and for manipulating attributes of the selected node. In the top left part, there is a viewport area showing the current render of the scene.

- **node** - Describes a node in a graph, however, nodes can be included in the pin as well. A node can contain multiple attributes and pins. There are several types of nodes available. A brief overview of node types can be found in the appendix Node Types.
- **attribute** - Contains a value of a node. A node has one or more attributes. The value of the attribute is stored in its child node, which is plain text. There are multiple attribute types available such as a single number, a triplet of numbers (vectors and colors), a string, etc.
- **pin** - A node type defining inputs in a node. In the picture, those are the circular areas in the upper area of each node. The pin can refer to a node in the scene graph, or contain one or more nodes. In the second case, the nodes are not visible in the node editor.
- **OCS2** - A scene node. This node contains the graph node.
- **graph** - A node starting the graph of a material. The graph node contains only the nodes of the **node** type. Those nodes can be seen directly in the graph area in the picture.

Chapter 4

Converter Specification

The converter is supposed to create a material that is not native to Unity, the Octane PBR Override material. A few steps were made to enable conversion to this material and to ensure the existence of its description in a file.

4.1 Reloading Octane Materials

This material has the issue that when being written to an existing material file, that file could not keep the change and return to its previous state. As written in section PBR Override material, the Octane is saved as an XML in the material file. This file is loaded when Octane starts operating and during testing was proven, that the material itself is being cached even during re-import of the material file or during the refreshing of the asset database. The solution to this can be seen in the chapter Converter Implementation and Functionality.

4.2 Conversion Tool

There are several ways how to implement the conversion tool. The tool can consist of a list of statements as is in the case of Wavefront format or Principled BRDF in Blender, or a graph structure similar to that in Octane.

The proposed method is a graph structure that copies the structure of Octane XML. There are several reasons.

1. The structure of the MTL format has a high probability of absence of several statements and there is a possibility that the order of the statements is mixed up.
2. Transferring the graph to an XML is easier than in the case of the list of statements.
3. Testing the correctness of the conversion tool had to be made as a conversion of one Octane material to another Octane material, which refers directly to the previous reason.

4.3 Material Reading

To make the conversion as efficient as possible, the materials have to be read directly from the file.

Because the Wavefront format is saved in the ASCII format only, reading it is straightforward. The only limitation is that Wavefront format enables omitting statements and their order is not directly specified.

In the case of the FBX format, there are two ways of saving a file, binary and ASCII versions. To read such files, an FBX reading tool is needed. There are many tools available:

- **IrrExt** [CSD15]
  An FBX loader and an extension to the Irrlicht Engine. Provides loading mesh and basic material properties. The project is written in C/C++, there would be a need to create a dynamic library in order for it to work in Unity.
- **Python FBX reader API** [BG18]
  An FBX reader written in Python. All known data types should be readable for FBX files from 2006-2012, files of later date were not tested. This reader is unable to directly read an ASCII FBX file. There would be a workaround for this reader to work in Unity.
- **fbx-conv** [ls20]
  A command line tool that uses FBX SDK to read the content of FBX

files. There are pre-compiled binaries available. The project is written in C/C++. A dynamic library would be needed in order for this tool to work in Unity.

- **openfbx** [ds13]
  C/C++ FBX importer capable of reading ASCII FBX files. This tool has a limited range of readable properties. A dynamic library would be needed for this tool to work in Unity.

- **FBX manipulation for .NET** [hms19]
  This tool can read and write both binary and ASCII FBX files. Format detection and advanced manipulation with FBX nodes are not available.

- **C++ Library for reading and writing FBX files** [js18]
  C++ FBX reading and writing tool. Supports FBX binary files. Allows inspection of FBX files in JSON format. A dynamic library would be needed for this tool to work in Unity.

- **OpenFBX** [ns21]
  FBX importer used by Flax Engine and Lumix Engine. The project is written in C/C++, so a dynamic library would be needed for this tool to work in Unity.

- **FbxWrapper** [Whi19]
  Fbx wrapper written in C/C++ for FBX SDK. Converts only certain parts of FBX SDK.

- **FBX SDK for Unity**
  FBX SDK provided by Autodesk for Unity. Guide for usage can be found here [Unia]. Reading the FBX file graph should follow the last line of code in the guide
  `importer.Import (scene);`
  which loads the content of the file to the scene variable and thus enables manipulation with the imported file.

Further issues contain the possible difference between materials that are from empirical and PBR renders. To solve this issue, a system that realizes the difference must be implemented, even if the process itself would depend on a user.

Chapter 5

Converter Implementation and Functionality

The converter itself is implemented in the way so saving the converted material as Octane XML would be as straightforward as possible.

For the implementation and testing was used this software:

- Unity 2019.3.5f1
- Blender 2.9.1
- Maya 2020
- Visual Studio 2019

Required packages and plugins are:

- Octane for Unity, version: 1.2.0.1410 or higher
- Autodesk FBX SDK, version: 4.0.1 or higher

Autodesk FBX SDK installation

If the Autodesk FBX SDK is already in the device, search for the manifest.json

file which is located in the location */ProjectFolder/Packages/manifest.json*. Open the file and add `"com.autodesk.fbx": "4.0.1",` to the list of dependencies. If there are further issues with the package installation, follow the manual posted on the Unity website [Uni21].

Octane for Unity installation

The Octane for Unity plugin can be found on the Unity Asset Store website [Oto]. After clicking on the *Open in Unity* button, Octane Render for Unity will be opened in the Package Manager. The *Import* button will add Octane to the project.

Installation of the converter

After adding the *MaterialConverter.unitypackage* to the project, import settings will be opened. After selecting files that will be imported, the converter will be installed. The converter requires both the Octane for Unity plugin and Autodesk FBX SDK to be installed.

There was a need for creating a new file. The reasons for such implementation were the fact that Octane materials were not responding to refreshing the asset database and to the re-importing of the material files. Due to those issues and later due to the fact that the methods provided by Octane need an existing material file in order to save the material properties, a new file for the converted result is to be made.

5.1 Converter Structure

The converter uses two classes for loading imported assets, the Scripted Importer [Unid] and the Asset Postprocessor [Unic].

The difference between those two classes is that that Scripted Imported enables importing files of unsupported formats to Unity. Those formats are distinguished by their file extension.

The Asset Postprocessor on the other hand enables developers to step into the importing process of files of the supported file formats. This way the original parameters of the imported files can be read before the start of the conversion to the inner format implemented in Unity.

The inner structure of the converter is based on the Octane material structure. This structure of Octane material is described for the scripting language

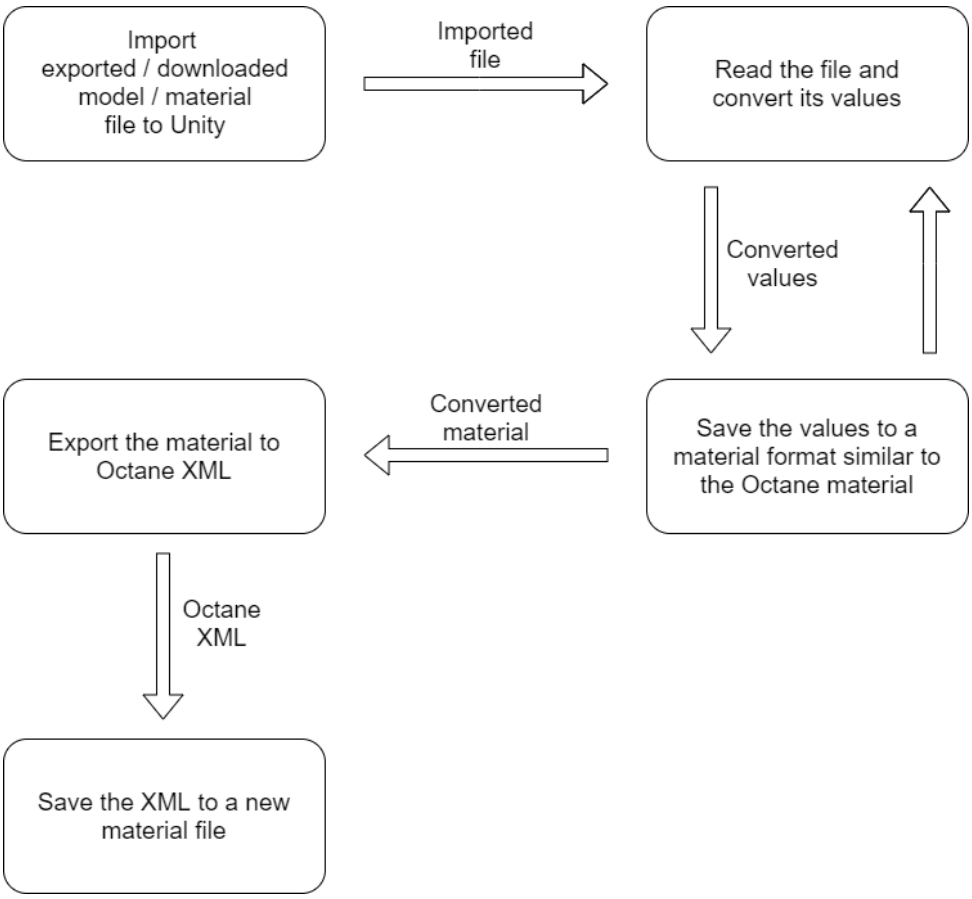


Figure 5.1: This diagram describes the way the converter works.

Lua [str13], however similar behavior is expected to work for C# as well. The final result is written to a material as an XML [Mic].

5.2 Wavefront Material Reading

As said in the Wavefront format description, the Wavefront format separates the contents of the model geometry and its materials into two file formats. The files with the extension OBJ are supported and when imported to Unity together with files with the file extension MTL, Unity creates a prefab of the OBJ file with converted materials from the MTL file applied. However, the files of the extension MTL are not supported by Unity themselves, only in the combination with the OBJ files. To read those files, a derived class of Scripted Importer must have been implemented.

Materials provided by Wavefront are described by a list of statements. There

are many statements and many operators, however, due to the fact that Wavefront materials are primarily formatted as an empirical model, some were omitted. Ambient color and the `illum` statements are examples of such cases. By observing the behavior of the Blender exporting tool, the Ambient color in the combination with the `illum` statement are used to save metallic values. However, such behavior will not be supported.

The supported statements and operators are:

- Supported statements:
  - `newmtl` - Indicates a new material.
  - `Kd` - Saved directly as an albedo color.
  - `Ks` - Saved directly as a specular color.
  - `Ke` - Saved directly as an emission color.
  - `Tf` - Saved directly as a transmission color.
  - `d` - Saved directly as an opacity value.
  - `Ni` - Saved directly as an IOR (Index Of Refraction).
  - `Ns` - Converted to a roughness. For this conversion, a conversion equation from Blender was used:

    $$roughness = 1 - \frac{\sqrt{\min(900, \texttt{Ns})}}{30}$$

    The equation can be found here [Mon19].
  - `map_Kd` - Saved directly as an albedo color map.
  - `map_Ks` - Saved directly as a specular color map.
  - `map_Ns` - Saved directly as a roughness map.
  - `map_d` - Saved directly as an opacity map.
  - `bump` - Saved directly as a bump map.
  - `map_Bump` - The same as the `bump` statement.
  - `disp` - Saved directly as a displacement map.
  - `map_Disp` - The same as the `disp` statement.
  - `map_Ke` - Saved directly as an emission color map.
  - `norm` - Saved directly as a normal map.
  - `map_Norm` - The same as the `norm` statement.
  - `decal` - By the notation from [DR95], the application of this statement should follow the equation

    $$result\_color = tex\_color(tv) * decal(tv) + mtl\_color * (1.0 - decal(tv))$$

for the ambient, diffuse and specular color. However, in this project was selected a different method. The method suggests that this statement affects the result of the albedo color, the specular color and roughness components and is inspired by the Figure 5.3. The resulted application of this statement to the previously mentioned material components is

$$component = \text{mix}(base\_color, \texttt{decal})$$

where *base_color* is albedo color, albedo texture, or their multiplication, and the ratio of this application is managed by a second copy of the texture (see example in Figure 5.2), which is loaded as a transparent texture. If there is not specified one of those three components, the default one will be used. The resulting effect is similar to the effect described here. More is available in the testing sections.

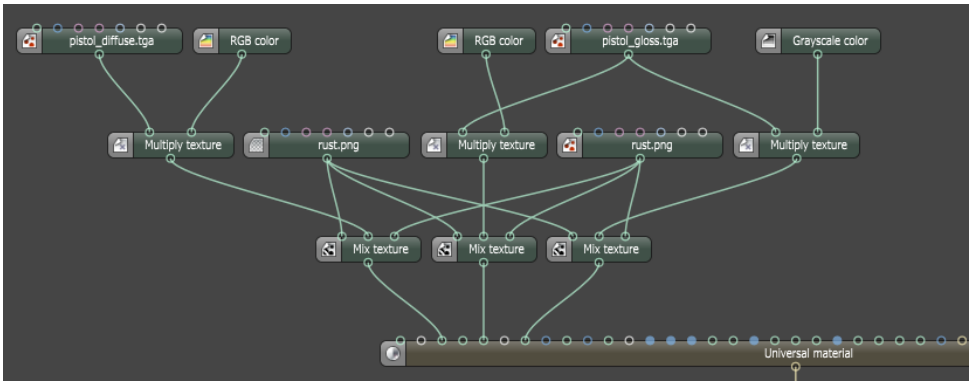


Figure 5.2: Application of the decal to the components of the material. The first connected component is the albedo color, the second is the specular color and the third is the roughness value. The decal used in this graph is the *rust* texture.

- `map_Decal` - The same as the `decal` statement.

- Supported operators (other operators will result in default value):
  - `xyz` - Converter converts color from the XYZ color space to RGB according to conversion matrix [Scr15].
  - `-bm` - This operator is followed by a value that indicates the bump multiplier. It is multiplied with the bump map.

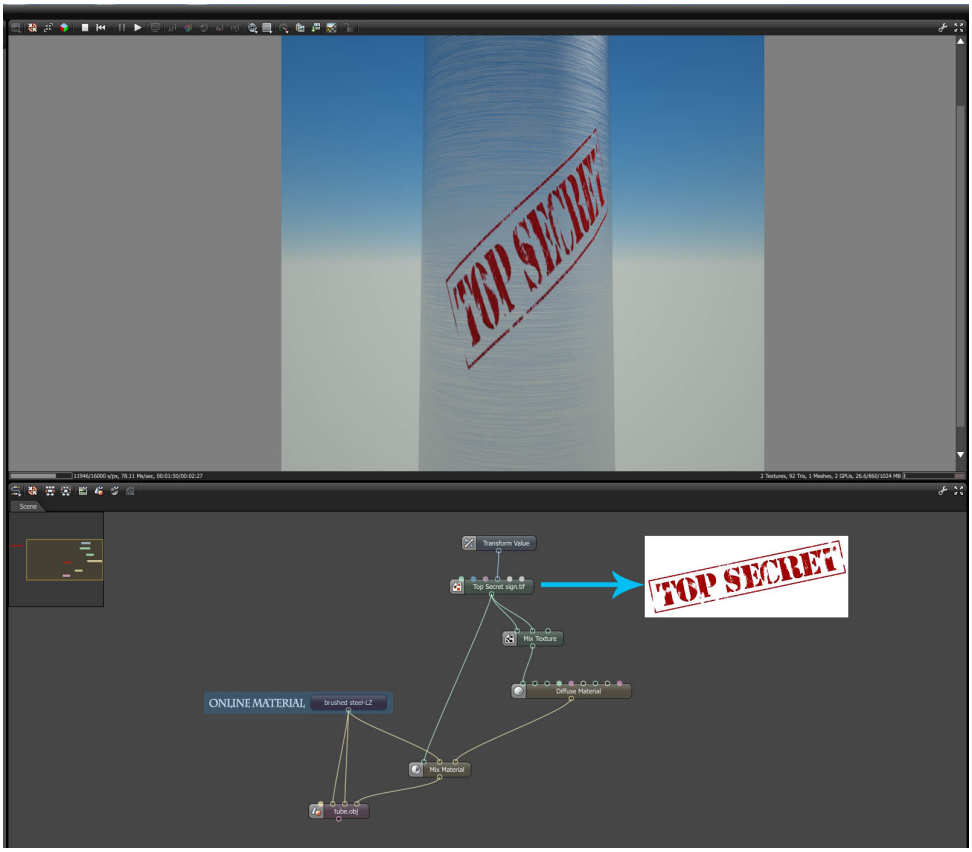


Figure 5.3: Graph describing the application of the decal texture to the material of the model [Jab14].

Maps and values of the same attribute type (e.g `Kd` and `map_Kd`) are multiplied.

If the file contains an error in the form of misspelled statement, that statement will be ignored and the rest of the material will be disposed of. This measure is created because of the possibility that the `newmtl` statement is misspelled and the converter would add the definition of the new material to the previous one. The converter expects that each statement would be written correctly up to one time per material definition.

5.3 FBX Reading

There were many possible tools available, however, the most promising one was the FBX SDK provided by Autodesk, mainly due to the fact that this

library is available natively in Unity. This package contains methods that enable browsing through the FBX file. This web page shows how to start reading the FBX file [Unia]. The reading of the FBX file should follow the scene import to a variable.
There can be limitations to some functions and a small mistake would create a difficult situation for the Unity developers. In order to avoid some difficulties, this thread in the Unity forum is recommended [vko21]. This thread shows a way how to unlock unavailable methods. However, this method is not used in this project.

Each FBX file can be read as a graph where each node can have multiple child nodes starting with the scene (root) node. Each node has several properties which contain node attributes and can contain source and destination objects as well. This behavior is applied to the properties as well. Iterating through all of the nodes and properties and objects connected to them would create an infinite cycle because the destination objects are primarily pointing to the root node.

The FBX file is read from the root node until materials are found, only the node children are searched. The materials are searched until the depth of three of their source objects. Information about textures can be found in this depth. The converted material values are as follows:

- Diffuse color - This color component is directly saved as an albedo color. Transparent color is immediately subtracted from this component. Similar behavior can be seen in Autodesk Maya.
- Displacement color - Directly saved as displacement color.
- Emission color - In FBX written as an "Emissive color". This component is directly saved as an emission color.
- Transparent color - This component is subtracted from the albedo color. The opacity is computed from this node and its factor as well. The color is multiplied by its factor and the opacity is computed as the average of the RGB channels from the result.
- Shininess - The value of this component can differ from which 3D tool the file was exported from. If the file was based on PBR rendering, the value is to be converted due to the equation

  $$roughness = 1 - (\sqrt{fbx\_shininess}/10)$$

  which is based on the equation from the Blender import tool [Mon18a]. If the 3D tool environment was based on an empirical reflection model, the computation is the same as for the `Ns` exponent from Wavefront materials .

- Specular color - The computation of the specular color differs for the files exported from the empirical and PBR-based 3D tools as well. While for the empirical conversion the application of the color is direct, for the PBR the specular component is converted as

  $$specular = fbx\_specular * 2.0$$

  which is based on the Blender import tool [Mon18b] and only the factor is used because the specular and diffuse colors are considered to be the same.
- Reflection color - This component is directly saved as metallic color.
- Normal map - This component is directly saved as a normal map.
- Bump map - This component is directly saved as a bump map.

Each of those properties can contain a texture attached as a source object. If such a situation arises, the texture and value are multiplied except the bump map and the normal map nodes. If there is a factor value related to each of those components, the components are multiplied with that factor before another operation starts.

5.4 Converter Usage

Before the conversion process starts, the Octane render must be loaded and rendering must be running. If not, the materials that are made by the converter might become empty. There is another condition that must be met and that is to import textures before the conversion starts. Several formats enable including a copy of textures into the file containing scene properties. Such behavior can be seen in the FBX files and some 3D tools enable including a texture copy to the FBX file. However, it is recommended to import the textures beforehand to the project.

To start the conversion, a file with the extension *.mtl* or *.fbx* must be imported to the Unity project. Then, all materials that are in the scene of the file are converted to the Octane format. The conversion can cause an error message *"The asset at path has been scheduled for reimport during the Refresh loop and Loading of it has been attempted. Doing this can lead to the AssetDatabase returning two versions of the same asset. Please ensure that code which attempts to reimport this asset does not run while the editor is Refreshing. You can do so by checking the value of EditorApplication.isUpdating."* to be displayed in Unity. Those messages can be ignored, creating a copy

of material is intentional behavior of the converter. However, after the conversions are finished, it is highly recommended to switch the converter off before any further processing of the imported model/scene or before doing any other activities that could cause model reimport.

The converter can be controlled by using a menu which can be found in the Octane section in the top panel, which can be seen in Figure 5.4. There the converter can be switched off/on, the converted formats available can be selected, FBX log can be enabled and the selection of whether the imported model is made in PBR or empirical environment. The log applies to the last converted FBX file and shows materials included in the file and their properties.

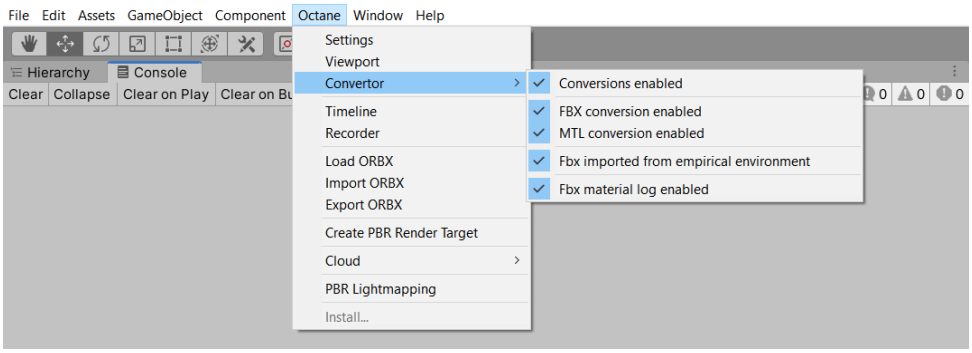


Figure 5.4: Menu used to operate the converter.

The converted materials are saved as new material files to the Assets/Materials/Converter folder. If such a path does not exist, the converter creates this destination folder.

The new materials can be further manually customized, see Figure 3.4.

Chapter 6

Testing

The testing was performed on several 3D models. The materials applied to those models were gradually changed to determine the functionality of the converter and to determine further progress.

6.1 Loading Material Files

This part of testing determined whether the goal of this project is achievable. The testing was performed on simple Octane materials whose color values were manually changed in a text editor.

The values were restored to their original value when the material was being applied to a model and the rendering was switched on.

If the material was already applied to the model, the values were changed and the re-importing of the material file was performed, the material did not change. The same behavior was observed when the asset database was refreshed.

Even though the material did change when the play button was clicked on, when the shader was changed to PBR Override, or when some of the scripts were saved, the reason for such behavior and the methods that started it were not found.

The last part of this testing was to create a new file and save a copy of another material to it. The result was an exact copy of the tested material file. Later, due to the difficulties connected with the Octane script IDs, the method was changed to only overwrite the Octane XML of the material file with methods that are available through the Octane for the Unity plugin.

6.2 Conversion Testing

Conversion from Octane material to Octane material

To test the correctness of the converter, the first conversion tests were directed towards the creation of an identical copy of the existing Octane material. Those tests were performed to determine the correctness of the inner conversion format and to determine its writing correctness.

Conversion from FBX and Wavefront materials to Octane materials

The converter was tested on several models due to the variation of available materials. The models are shader ball [Mak15], a tree model [ztr], a pistol model [alp] and basic geometry such as a plane.

Figure 6.1: Comparison between application of the original decal equation (on the left) and the new version of the application of the decals (on the right).

Due to the testing, the decals were the one Wavefront statements which application was entirely changed. The final result of such conversion can be seen in Figure 6.1 in comparison with the result of the original method. Its application process is described in the implementation part.

The decal texture that was used is a rust texture [DTr].

Figure 6.2: Usage of an image map used for displacement (on the left), which was the original usage, and for normals (on the right).

There exist errors that can be brought into the scene with the 3D models, such as wrongly assigned texture. Those cases can result in a difference of the required and the acquired shape and appearance of the model. Such a case can be seen in Figure 6.2 where the originally written material contained a statement used for the displacement texture map instead of the normal texture map.

Some tests were created in order to determine whether the converter can

Figure 6.3: Test of multiple materials converted from one file.

convert multiple materials from one file. The result of multiple materials being converted from one file can be seen in Figure 6.3, where different materials are used for the leaves and for the trunk and branches.

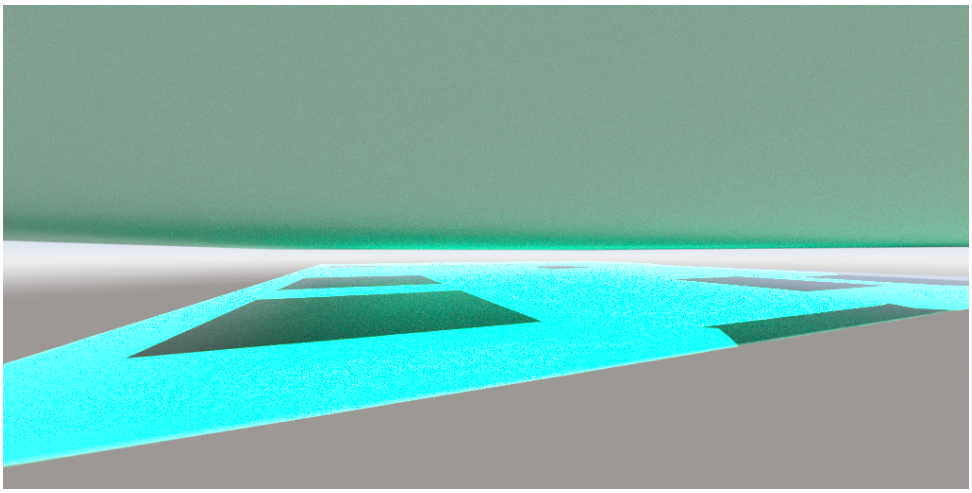


Figure 6.4: Test of emission values.

This test in Figure 6.4 was performed on a Wavefront material that contained emission values only. The emission consisted of a crate texture and a turquoise color. The result was their mixture by multiplication.

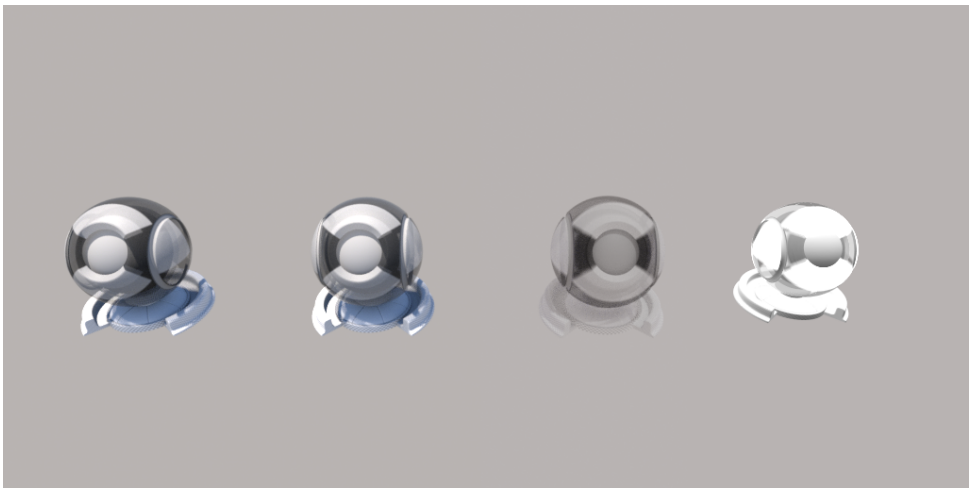


Figure 6.5: Comparison of transparency of the converted material from Blender (on the left), converted material from Maya (in the middle on the left), material from Blender (in the middle on the right) and material from Maya (on the right).

By using the shader balls in Figure 6.5, the materials with the transparency value of 0.5 are being compared. Each of those materials has a black-white

checker texture applied. The first two shader balls from the right are rendered in Unity by using the Octane render, the third shader ball is rendered in Blender Eevee render and the fourth one in the Maya Software render.

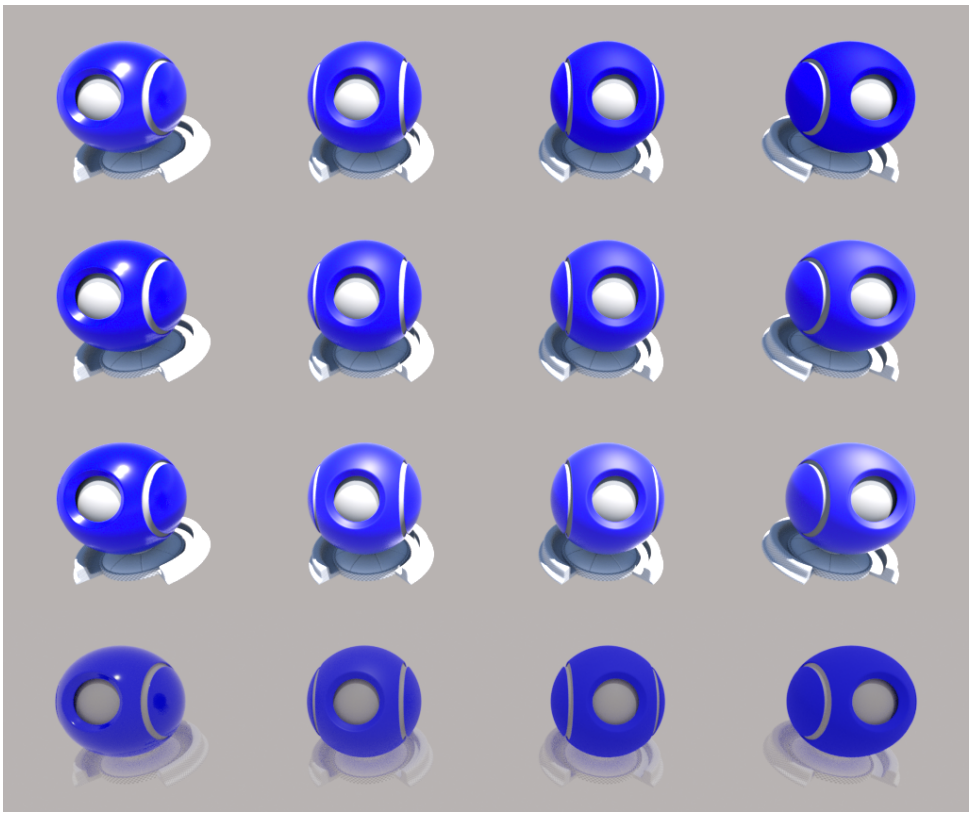


Figure 6.6: Comparison of materials between Unity standard material (top row), converted material (second row), a material created manually in Octane (third row) and Blender Principled BSDF material (fourth row).

The Figure 6.6 compares materials with different roughness values (smoothness for the first row, which is computed as $1 - roughness$). The roughness values are from the left to the right equal to 0.118, 0.529, 0.817 and 0.918. Their original specular exponent values are 700, 200, 30 and 6.

The top three rows are rendered in Unity by the Octane render, the last row is rendered in Blender by the Eevee render. Due to the size of the image, the renders of the materials from Maya and their conversions are placed in the next image.

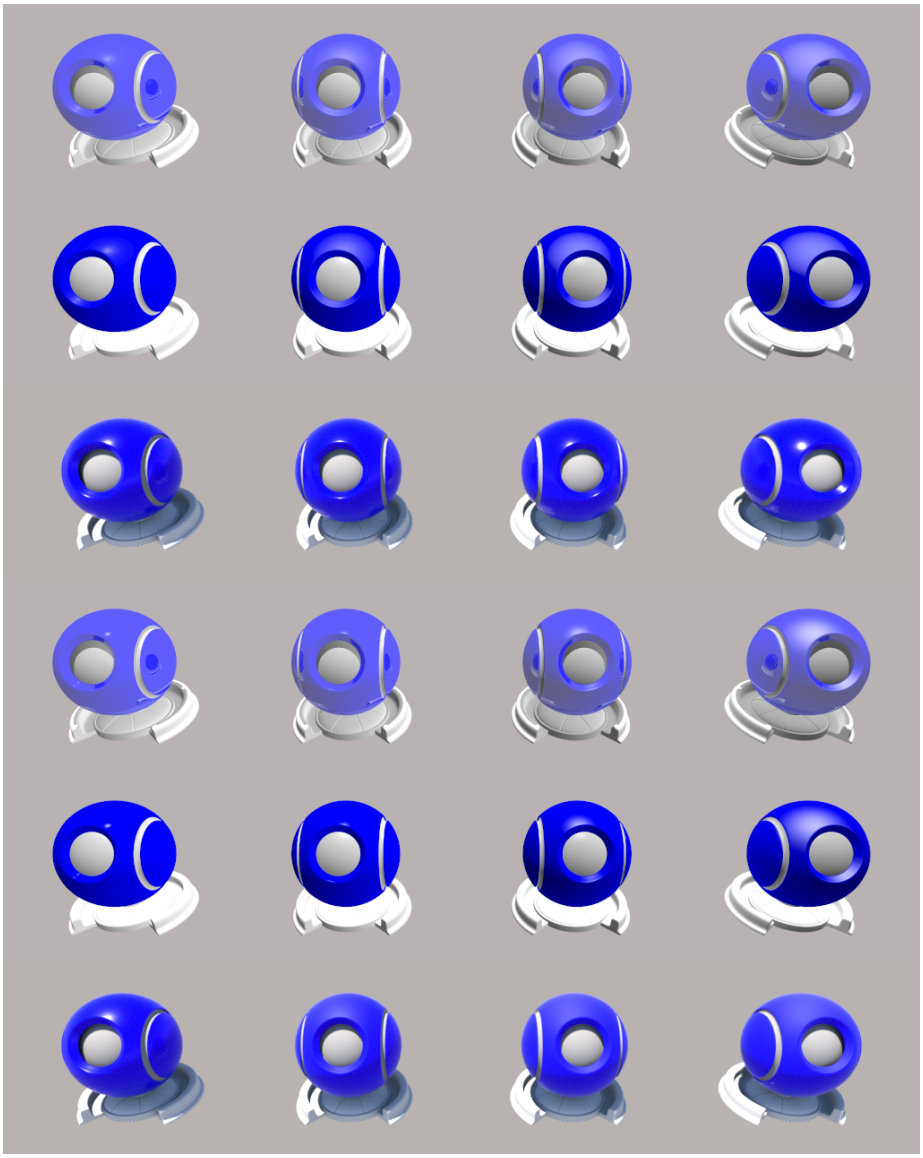


Figure 6.7: Comparison of materials between Maya PhongE material with raytracing enabled (first row) and disabled (second row), their conversion in Octane (third row), the Maya Phong material with raytracing enabled (fourth row) and disabled (fifth row) and their conversion in Octane (sixth row).

The comparison in Figure 6.7 is between materials created in Autodesk Maya with their converted alternatives in Octane. The roughness values remain the same as in the previous comparison, although the roughness values were not exported with Maya PhongE materials. The converted materials were rendered in Unity by the Octane render and those from Maya by the Maya Software render. In Maya ambient light was used as well. The process of computing roughness values is described in the implementation part.

Chapter 7

Conclusions

This thesis analyzed and described Octane and its material shader. The FBX format and the Wavefront format, which were used as the input formats for the conversion to the Octane PBR Override material, were described as well.

The converter was designed, implemented and its functionality was proven with the testing of materials, which followed the implementation. Although the converted materials were not entirely the same as their original, their similarity was close enough to consider the conversion successful. The new converted materials share the same folder, which is the *Assets/Materials/Converter* folder, their name is supplemented by a prefix *new_* and their shader is changed to the PBR Override shader provided by Octane.

This conversion tool could be expanded on other 3D data formats in the future by using the implemented methods in the converter, which are as general as possible.

Because of its inability to assign the new materials to the models, the converter can be combined with scripts and prefabs that assign materials by their name, shader, or both to the 3D models.

Appendix A

Bibliography

[IPI13] *Illumination 1: The Phong Illumination Model*, Department of Computer Science & The Oden Institute of Computational Engineering and Sciences, 2013, Last visit: May 2021, [Lecture]. Available at https://www.cs.utexas.edu/~bajaj/graphics2012/cs354/lectures/lect14.pdf.

[WMT20] Wavefront material template library (mtl) file format, February 14 2020,
Last visit: May 2021 [Online]. Available at https://www.loc.gov/preservation/digital/formats/fdd/fdd000508.shtml.

[ds13] DOBKERATOPS & SPOL., *openfbx*, GitHub, March 13 2013,
Last visit: May 2021, GNU LESSER GENERAL PUBLIC LICENSE Version 3, 29 June 2007, [Online project]. Available at https://github.com/dobkeratops/openfbx.

[js18] JSKOREPA & SPOL., *C++ Library for reading and writing FBX files*, GitHub, August 7 2018,
Last visit: May 2021, MIT License Copyright (c) 2017 Jakub Skořepa, [Online project]. Available at https://github.com/jskorepa/fbx/tree/master.

[hms19] HAMISH-MILNE & SPOL., *FBX manipulation for .NET*, GitHub, July 31 2019,
Last visit: May 2021, GPL license - GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007 / LGP license - GNU LESSER GENERAL PUBLIC LICENSE Version 3, 29 June 2007, [Online

project]. Available at https://github.com/hamish-milne/FbxWriter.

[ls20] LIBGDX & SPOL., *fbx-conv*, GitHub, March 8 2020, Last visit: May 2021, Apache License Version 2.0, January 2004, [Online project]. Available at https://github.com/libgdx/fbx-conv.

[ns21] NEM0 & SPOL., *OpenFBX*, GitHub, May 9 2021, Last visit: May 2021, MIT License Copyright (c) 2017 Mikulas Florek, [Online project]. Available at https://github.com/nem0/OpenFBX.

[Aut] AUTODESK, *Autodesk help: FBX Scenes*, Last visit: May 2021, Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License, [Manual]. Available at https://help.autodesk.com/view/FBX/2017/ENU/?guid=__files_GUID_4F644045_380D_4B75_A2A3_D39DDE53BEDD_htm.

[BG18] C. BARTON and A. GESSLER, *Python FBX reader API*, January 24 2018, Last visit: May 2021, GNU General Public License v2.0, [Online project]. Available at https://github.com/ideasman42/pyfbx_i42.

[Bli77] J. F. BLINN, Models of light reflection for computer synthesized pictures, *SIGGRAPH Comput. Graph.* **11** no. 2 (1977), 192–198, [Article]. https://doi.org/10.1145/965141.563893.

[Bri10] A. BRITO, Octane render: Fully gpu unbiased render engine, (2010), Last visit: May 2021, [Article]. Available at https://www.blender3darchitect.com/light-and-rendering/octane-render-fully-gpu-unibased-render-engine/.

[CSD15] F. CHRISTIAN STEHNO, CASWAL "ZEUSSY" PARKER and G. DAVIDSON, *IrrExt*, Sourceforge, October 3 2015, Last visit: May 2021, [Online project]. Available at https://sourceforge.net/p/irrext/code/91/tree/trunk/extensions/scene/IMeshLoader/fbx/.

[DR95] L. T. DIANE RAMEY, LINDA ROSE, *MTL material format (Lightwave, OBJ)*, 4,2 ed., October 1995, Last visit: May 2021, Copyright 1995 Alias|Wavefront, Inc. Available at http://paulbourke.net/dataformats/mtl/.

[Eck18] D. J. ECK, *Introduction to Computer Graphics*, **1.2**, Hobart and William Smith Colleges, January 2018, Section 4.1 Introduction to Lighting. Available at http://math.hws.edu/graphicsbook/c4/s1.html.

[Hou] B. Houston, Ben houston's ultimate guide to 3d file formats, Last visit: May 2021, COPYRIGHT © 2021 THREEKIT INC. Available at https://www.threekit.com/blog/when-should-you-use-fbx-3d-file-format.

[Hou15] B. Houston, Extending wavefront mtl for physically-based rendering, (2015). Available at http://exocortex.com/blog/extending_wavefront_mtl_to_support_pbr.

[JrF05] J. S. Jiří Žára, Bedřich Beneš and P. Felkel, *Moderní počítačová grafika*, **2**, pp. 319–336, Computer Press, 2005, ISBN: 80-251-0454-0. Available at https://dcgi.fel.cvut.cz/cgg/ModerniPocitacovaGrafika/kniha.pdf.

[Kop14] S. J. Koppal, *Lambertian Reflectance*, computer vision: a reference guide ed., pp. 441–443, Springer US, Boston, MA, 2014, ISBN: 978-0-387-31439-6. https://doi.org/10.1007/978-0-387-31439-6_534.

[KE09] M. Kurt and D. Edwards, A survey of brdf models for computer graphics, *SIGGRAPH Comput. Graph.* **43** no. 2 (2009), ISSN: 0097-8930, Last visit: May 2021. https://doi.org/10.1145/1629216.1629222.

[Mic] Microsoft, *XmlDocument class*, Last visit: May 2021, [Manual]. Available at https://docs.microsoft.com/en-us/dotnet/api/system.xml.xmldocument?redirectedfrom=MSDN&view=net-5.0.

[Mon18a] B. Montagne, Roughness conversion in Blender for FBX file format, October 16 2018, commit: b3257c11365e, shininess, from import_fbx.py, line 1340, Last visit: May 2021, [Online project]. Available at https://developer.blender.org/rBAb3257c11365e38436a86a73cf42a300a305c33aa.

[Mon18b] B. Montagne, Specular color conversion in Blender for FBX file format, October 13 2018, commit: 94a1268efa6a, specular, from import_fbx.py, line 1335, Last visit: May 2021, [Online project]. Available at https://developer.blender.org/rBA94a1268efa6a16b2e85b95a6ad145e3bbd63f018.

[Mon19] B. Montagne, Blender developer's page, import of the OBJ files, July 30 2019, commit: 27381001d7b9, Ns exponent, line 361, Last visit: May 2021, [Online project]. Available at https://developer.blender.org/rBA27381001d7b9332eb669f1023f14b40d1f15f962.

[MU12] R. Montes and C. Ureña, *An Overview of BRDF Models*, Tech. report, Dept. Lenguajes y Sistemas Informáticos University of Granada, February 2012, pp. 6–7, Last visit: May

2021. Available at https://digibug.ugr.es/bitstream/handle/10481/19751/rmontes_LSI-2012-001TR.pdf;jsessionid=900B79928FE698AF941A00F265A6EF0B?sequence=1.

[Nic16] C. Nichols, The truth about unbiased rendering, (2016), Last visit: May 2021. Available at https://www.chaosgroup.com/blog/the-truth-about-unbiased-rendering.

[Oto] Otoy Inc., *Octane documentation: Unity® Standard Material*, Last visit: May 2021, [Manual]. Available at https://docs.otoy.com/UnityH/UnityManual.htm#Unity/UnityStandardMaterial.htm.

[PJH16] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*, 3 ed., Elsevier, September 30 2016, ISBN: 978-0-12-800645-0, Section 8.4 Microfacet Theory. Available at https://www.pbr-book.org/3ed-2018/Reflection_Models/Microfacet_Models.

[Scr15] Scratchpixel, Introduction to light, color and color space, October 8 2015, Conversion table CIE XYZ to sRGB, Last visit: May 2021, [Lecture]. Available at https://www.scratchapixel.com/lessons/digital-imaging/colors/color-space.

[str13] stratified, Nodes and nodegraphs in lua, *Lua Scripting* (2013), Last visit: May 2021. Available at https://render.otoy.com/forum/viewtopic.php?f=73&t=37333.

[TON13] TON, Fbx binary file format specification, *Blender Developers Blog* (2013), Last visit: May 2021. Available at https://code.blender.org/2013/08/fbx-binary-file-format-specification/.

[TJL] R. Tunnel, J. Jaggo, and M. Luik, Computer graphics learning materials, Last visit: May 2021. Available at https://cglearn.eu/pub/advanced-computer-graphics/physically-based-shading.

[Unia] Unity Technologies, *FBX SDK Developer's Guide*, Last visit: May 2021, [Manual]. Available at https://docs.unity3d.com/Packages/com.unity.formats.fbx@2.0/manual/devguide.html.

[Unib] Unity Technologies, *Unity Documentation: 3D formats*, Last visit: May 2021, [Manual]. Available at https://docs.unity3d.com/560/Documentation/Manual/3D-formats.html.

[Unic] Unity Technologies, *Unity Documentation: Asset Postprocessor*, Last visit: May 2021, [Manual]. Available at https://docs.unity3d.com/ScriptReference/AssetPostprocessor.html.

[Unid] Unity Technologies, *Unity Documentation: Scripted Importers*, Last visit: May 2021, [Manual]. Available at https://docs.unity3d.com/Manual/ScriptedImporters.html.

[Uni21] Unity Technologies, *About Autodesk® FBX® SDK for Unity*, March 12 2021, Last visit: May 2021, [Manual]. Available at https://docs.unity3d.com/Packages/com.autodesk.fbx@4.0/manual/index.html.

[vko21] vkovec, Enabling fbx sdk methods, April 12 2021, post 20, Last visit: May 2021, [Online]. Available at https://forum.unity.com/threads/autodesk-fbx-build-errors-for-windows.716867/.

[Whi19] J. While, *FbxWrapper*, Bitbucket, January 6 2019, Last visit: May 2021, [Online project]. Available at https://bitbucket.org/johnwhile/fbxwrapper/src/master/.

Appendix B

References of the External Files

[srcDTr] Dirt transparent rust - rusty roblox. Available at https://www.pngkey.com/detail/u2q8a9u2i1a9u2r5_dirt-transparent-rust-rusty-roblox/.

[srcalp] ALPENWOLF, Colt python 8 inch free low-poly 3d model. Available at https://www.cgtrader.com/free-3d-models/military/gun/colt-python-8-inch.

[srcJab14] JABERWOCKY, Applying decals within octane, Sep 2014. Available at https://render.otoy.com/forum/download/file.php?id=36351&mode=view.

[srcKim] M. KIM, Vectors used in reflection models, Dept. of Computer Science University of Seoul. Available at http://www.minho-kim.com/courses/13fa71033/data/redbook-07.pdf.

[srcMak15] M. MAKIN, Shader ball, Mar 2015. Available at https://github.com/derkreature/ShaderBall.

[srcOto] I. OTOY, Octane render for unity. Available at https://assetstore.unity.com/packages/tools/integration/octanerender-for-unity-installer-scene-105646.

[srcRai] RAINWARRIOR COMMONSWIKI(BASED ON COPYRIGHT CLAIMS), Comparison between blinn-phong and phong reflection models, CC BY-SA 3.0. Available at https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model#/media/File:Blinn_phong_comparison.png.

[srcSmi] B. Smith, Physically based rendering: From theory to implementation, CC BY-SA 3.0. Available at https://commons.wikimedia.org/w/index.php?curid=1030364.

[srcztr] ztrztr, midpoly tree pack 002 free 3d model. Available at https://www.cgtrader.com/free-3d-models/exterior/landscape/midpoly-tree-pack-002.

Appendix C

Octane Material Node Types

NT_UNKNOWN	0	NT_TEX_FLOAT	31
NT_GEO_MESH	1	NT_TEX_GAUSSIANSPECTRUM	32
NT_MAT_MAP	2	NT_TEX_RGB	33
NT_GEO_GROUP	3	NT_TEX_IMAGE	34
NT_GEO_PLACEMENT	4	NT_TEX_ALPHAIMAGE	35
NT_GEO_SCATTER	5	NT_TEX_FLOATIMAGE	36
NT_FLOAT	6	NT_ENV_TEXTURE	37
_NT_FLOAT2	7	NT_TEX_MIX	38
_NT_FLOAT3	8	NT_TEX_MULTIPLY	39
NT_INT	9	NT_TEX_COSINEMIX	40
_NT_INT2	10	NT_TEX_CLAMP	41
NT_BOOL	11	NT_TEX_SAWWAVE	42
NT_IMAGE_RESOLUTION	12	NT_TEX_TRIANGLEWAVE	43
NT_CAM_THINLENS	13	NT_TEX_SINEWAVE	44
NT_ENV_DAYLIGHT	14	NT_TEX_CHECKS	45
NT_IMAGER_CAMERA	15	NT_TEX_INVERT	46
NT_MAT_GLOSSY	16	NT_TEX_MARBLE	47
NT_MAT_DIFFUSE	17	NT_TEX_RGFRACTAL	48
NT_MAT_SPECULAR	18	NT_TEX_GRADIENT	49
NT_MAT_MIX	19	NT_TEX_FALLOFF	50
NT_MAT_PORTAL	20	NT_TEX_COLORCORRECTION	51
_NT_CAMERARESPONSE	21	NT_EMIS_BLACKBODY	53
NT_TEX_TURBULENCE	22	NT_EMIS_TEXTURE	54
NT_KERN_PMC	23	NT_LOCAL_APP_PREFS	55
NT_KERN_DIRECTLIGHTING	24	NT_RENDERTARGET	56
NT_KERN_PATHTRACING	25	NT_ENUM	57
NT_KERN_INFO	26	NT_MED_ABSORPTION	58
NT_TRANSFORM_3D	27	NT_MED_SCATTERING	59
NT_TRANSFORM_SCALE	28	NT_PHASE_SCHLICK	60
NT_TRANSFORM_ROTATION	29	NT_POSTPROCESSING	61
NT_SUN_DIRECTION	30	NT_CAM_PANORAMIC	62

NT_TEX_DIRT	63	NT_CAM_BAKING	94
NT_OBJECTLAYER_MAP	64	NT_VOLUME_RAMP	95
NT_OBJECTLAYER	65	NT_VERTEX_DISPLACEMENT	97
NT_TRANSFORM_2D	66	NT_MED_VOLUME	98
NT_TRANSFORM_VALUE	67	NT_ANIMATION_SETTINGS	99
NT_ANNOTATION	68	NT_FILM_SETTINGS	100
NT_KERN_MATPREVIEW	69	NT_DIRECTORY	101
NT_PROJECT_SETTINGS	70	NT_GEO_JOINT	102
_NT_REMOTE_APP_PREFS	71	NT_LUT_CUSTOM	103
NT_SPLIT_PANE	72	NT_TEX_W	104
NT_WORK_PANE	73	NT_RENDER_JOB_GROUP	105
NT_PROJ_CYLINDRICAL	74	NT_TEX_ADD	106
NT_PROJ_LINEAR	75	NT_TEX_COMPARE	107
NT_PROJ_PERSPECTIVE	76	NT_TEX_SUBTRACT	108
NT_PROJ_SPHERICAL	77	NT_TEX_TRIPLANAR	109
NT_PROJ_UVW	78	NT_GEO_PLANE	110
NT_PROJ_BOX	79	NT_PROJ_TRIPLANAR	111
NT_DISPLACEMENT	80	NT_IMPORT_IMAGE_PREFS	112
NT_TEX_RANDOMCOLOR	81	NT_TEX_INSTANCE_COLOR	113
NT_IMPORT_ALEMBIC_PREFS	82	NT_TEX_INSTANCE_RANGE	114
NT_IMPORT_OBJ_PREFS	83	NT_TEX_BAKED_IMAGE	115
NT_STRING	84	NT_MAT_OSL	116
NT_PROGRAMMABLE_GRAPH_INPUT	85	NT_TEX_OSL	117
NT_RENDER_PASSES	86	NT_TEX_UVW_TRANSFORM	118
NT_TEX_NOISE	87	NT_IMPORT_FBX_PREFS	119
NT_FILE	88	NT_MAT_METAL	120
NT_TEX_SIDE	89	NT_MAT_TOON	121
NT_RENDER_LAYER	90	NT_TOON_RAMP	122
NT_GEO_VOLUME	91	NT_TOON_POINT_LIGHT	123
NT_ORC_VERSION	92	NT_TOON_DIRECTIONAL_LIGHT	124
NT_IMPORT_VDB_PREFS	93	NT_PROJ_OSL	125

NT_CAM_OSL	126
NT_PROJ_OSL_UV	127
NT_CAM_OSL_BAKING	128
NT_ENV_PLANETARY	129