

New Standards in Real-Time Graphics

A Physically-Based Graphics Pipeline

Joachim Lindkvist
2014

Bachelor of Fine Arts
Computer Graphic Arts

Luleå University of Technology
Institutionen för konst, kommunikation och lärande



New Standards in Real-Time Graphics

A Physically-Based Graphics Pipeline

Thesis in Computer Graphics

Joachim Lindkvist

Luleå Tekniska Universitet, Campus Skellefteå, 2014
Examensarbete 15 HP

Preface

This thesis concludes my bachelor's degree in Computer Graphics at the Luleå University of Technology. I want to sincerely thank my fellow students, family members and teachers for all of your support and guidance. Finally, I want to thank Virtual Basement for giving me all the tools, support and patience I could've possibly asked for.

Thank you all sincerely,
Joachim Lindkvist

Sammanfattning

I den här rapporten utvärderar jag vilka effekter *Physically-Based Rendering* får på den grafiska pipeline i spel-utveckling.

Undersökningsansatsen för den här rapporten består av två delar; litteraturstudien är delvis baserad på publikationer från SIGGRAPH konferensen (2013), vilket representerar de senaste utvecklingarna i spel industrin (bland annat). Syftet med litteraturstudien är att utvärdera hur spel industrin antar och implementerar principerna av *physically-based rendering*. Jag har också undersökt vilken teori och vilka mål som är grundläggande för *physically-based rendering* i spel.

Utvecklingen har också haft påverkan på vad som är kallat *shading* och *material* modellerna. Min arbetsinsats varit att undersöka de tidigare etablerade modellerna och använda de här som ett etablerat *benchmark* vid jämförelse.

Den andra delen av min undersökningsansats består av en empirisk undersökning. Den empiriska undersökningen i den här rapporten är en studie på den grafiska pipeline, mer specifikt så har jag genomfört en hel grafisk produktion på ett spel-objekt. Genom den här produktionen har jag utvärderat de nya utvecklingarna som tillkommit texturering pipeline. Texturering pipeline har utvecklats. Texturer som en grafiker är förväntad att skapa är annorlunda för en *material modell* som principiellt är *physically-based*. För att analysera vad den nya material modellen innebär har jag använt de äldre och etablerade modellerna som ett benchmark.

Abstract

In this paper I evaluate the impacts of *Physically-Based Rendering* on the graphics pipeline in game development.

The research approach for this paper is split into two parts; the literature study consists in part of publications from the SIGGRAPH conference (2013), which represents the latest developments in the industry. The purpose of the literature study is to examine how the games industry have adopted, and then implemented the principles of *physically-based rendering*. I have also examined what theory and goals are foundational to *physically-based rendering* in games.

The developments in rendering principles creates differences in, what's called the shading models and material models. To examine these new developments my approach is to examine the old shading models, and refer to these as the established benchmark.

The second part of my approach involves empirical research. The empirical research consists of a study on the graphics pipeline, specifically, I have gone through the entire production of a single graphical game asset, and in doing so evaluated the new developments to the texturing pipeline. The texturing pipeline has evolved, the material model is the culprit and the texture assets that an artist is expected to create is different for a physically principled material model. To analyze the new *material model* I have used the older and established models as the benchmark.

Table of Contents

1	Introduction	3
1.1	Background	3
1.2	Question Formulation.....	3
1.3	Purpose	3
1.4	Limitations.....	3
2	Glossary	4
3	Methodology.....	5
3.1	Research Approach	5
3.1.1	<i>Literature Study</i>	5
3.1.2	<i>Benchmarking</i>	5
3.1.3	<i>Empirical Research</i>	5
3.2	Method Critique.....	6
4	Theory.....	6
4.1	The Graphics Rendering Pipeline.....	6
4.1.1	<i>Architecture</i>	6
4.1.2	<i>The Geometry Stage</i>	7
4.1.3	<i>The Vertex Shader</i>	7
4.1.4	<i>The Rasterizer</i>	7
4.1.5	<i>Programmable Shading</i>	7
4.1.6	<i>The Pixel Shader</i>	7
4.1.7	<i>Languages</i>	8
4.2	The Shading Model.....	8
4.2.1	<i>Material Model</i>	8
4.2.2	<i>BRDF Terms</i>	8
4.2.3	<i>Diffuse Term</i>	8
4.2.4	<i>Specular Term</i>	8
4.2.5	<i>Fresnel Term</i>	9
4.2.6	<i>Ambient Term</i>	9
4.2.7	<i>Phong Reflectance</i>	9
4.3	Physically Based Rendering	9
4.3.1	<i>Goals of PBR</i>	9
4.3.2	<i>Industry adoption</i>	10
4.3.3	<i>Measured Materials</i>	10
4.3.4	<i>The Physically-Based Shading Model</i>	11
4.3.5	<i>Unreal Engine 4 Material Model</i>	11
5	Empirical Research	13
5.1	Asset Creation	13
5.2	Texturing for Unreal Engine 4	13
5.2.1	<i>Procedural Texturing with DDO</i>	13
5.2.2	<i>Iron</i>	13
5.2.3	<i>Plastic</i>	14
5.3	BRDF Implementation	15
5.3.1	<i>Lambert BRDF</i>	15
5.3.2	<i>Fresnel BRDF</i>	15
5.3.3	<i>Phong Specular BRDF</i>	15
6	Results	16
6.1	A new graphics pipeline.....	16
6.2	Industry Adoption	16

7	Discussion	17
7.1	The shading model	17
7.2	The material model.....	17
7.3	The texturing pipeline	18
8	Conclusions	19
8.1	Physically-Based Shading.....	19
8.2	Future Developments.....	19
9	References	20

1 Introduction

1.1 Background

New technologies and tools change the way we work. To stay competitive and relevant we have to keep learning new things. During this project, I will work with Virtual Basement LLC on their new unreleased Unreal Engine 4 game.

I will examine the new standards to the rendering and content-creation pipeline. This includes concepts such as Physically-Based Shading, Energy Conservation and image-based lighting. I will look at how these concepts gives us a new Material Model and compare it to the old and established. I will evaluate how the artists' work have changed with the new Material Model. Finally, I will look at how the games industry have adopted these principles.

1.2 Question Formulation

- ❖ How does the implementation of physically-based rendering impact the graphics pipeline?

1.3 Purpose

Physically-Based Rendering (PBR) for games is a very broad topic. I mainly cover the details that concern the graphical artist. I have listed the main objectives of this report, with that in mind.

This report should:

- Evaluate how PBR impacts the graphics pipeline.
- Evaluate how the video games industry have adopted it.

1.4 Limitations

This paper will not cover in-depth analysis of mathematical algorithms or implementations. I don't include theory on physics and radiometry, this is a well-documented area that's beyond the purpose of this paper.

2 Glossary

Shader, is a computer program that models light behavior on a 3D surface.

Specular, (Specular Term) refers to surface reflectance.

Diffuse, refers to body reflectance.

Fresnel, a model for angle-dependent reflectivity.

Vertex/Vertices, is a geometric point with a Position and Color.

SSS, (Sub-Surface Scattering), is a lighting model for translucency, commonly used for skin-shading.

Triangle, is the smallest geometric surface with three vertices/edges.

Vector3, a 3D vector with x, y, z-coordinates.

PBR, Physically-Based Rendering, generally refers to principles that are more physically accurate than previous generations.

BRDF, (Bi-directional Reflectance Distribution Function) “Defines how light is reflected at an opaque surface.” (Wikipedia, n.d.)

SVBRDF, (Spatially Varying Bi-Directional Reflectance Distribution Function)

Lambert, is the most common diffuse lighting model.

Phong, is a common specular lighting model.

Polygon, is a geometric surface with an undefined number of edges.

Edge, is a geometric line that consist of two connected vertices.

Normal, a vector3 that is perpendicular to a surface.

Tangent, a vector3 that is perpendicular to the Normal.

3 Methodology

In this section I will present the different methods I used to fulfill the stated purpose of this paper.

3.1 Research Approach

This paper is comprised of two parts: a literature study and an empirical study.

3.1.1 Literature Study

The subject of this report is fairly well documented but there are some areas that have had fairly recent developments. I have used the latest publications and newest research where I could find it. Publications from SIGGRAPH (2013) have provided me with a window into some of the latest developments and research papers available on the topics of this paper.

There are however some baseline *science* that hasn't changed a lot. In this case I have used comprehensive and commonly referenced material. One of the main reference sources is 'Real-Time Rendering' (Akenine-Möller, et al., 2008).

The publications from SIGGRAPH (2013) are often as far as I can tell at least informally peer-reviewed (Karis, 2013, p. 19) and written by members of the video games or visual effects industries. These publications are also presented at the conference (SIGGRAPH) to other professionals, which I think increase their validity.

I have also deemed some websites suitable to be used as sources for reference, such as the blog by Sébastien Legarde. The material on his blog has been widely referenced by established companies (Epic Games) and professionals of the video games industry. Whenever possible, I have also cross-referenced statements to assess their validity.

For definitions of common words and terminology I rely on Wikipedia as my main source. I judge *Wikipedia* to be well supervised and reviewed information.

3.1.2 Benchmarking

In this paper, my focus is on the Physically-Based principles that Epic Games implemented in Unreal Engine 4.

3.1.3 Empirical Research

The empirical research I will conduct will give me first-hand experience to draw from when gathering and choosing reference material. The literature study will in turn help me analyze the result of the empirical research.

3.1.3.1 Graphics Pipeline

Working with Virtual Basement, I have been given the opportunity to evaluate Unreal Engine 4's new physically-based graphics pipeline. The empirical research will consist of a study of the texturing pipeline and of the implementation of common shading¹ terms.

¹ Shading refers to the process of altering a 3D objects color using data such as *light angle* and *intensity* to try to emulate real world visual phenomenon. (Wikipedia, n.d.)

3.1.3.2 The texturing pipeline

I will go through the entire texturing pipeline for a single 3D game asset. The purpose of this is to study the new material model implemented by Epic in Unreal Engine 4.

3.1.3.3 Implementation of BRDF Models

I will look at the implementation of some common BRDF² terms. This will provide a benchmark for comparison when I examine the *new* BRDF Models that are more *physically-based*.

3.2 Method Critique

There are more I wish I could cover, but for the sake of expediency I had to cut some potential chapters from this paper. I would have liked to cross-examine and compare at least two differing implementations of Physically-Based Rendering. I am also aware that I do not have deep rendering knowledge to truly begin to evaluate the shading calculations.

4 Theory

In this section I will outline the architecture of the graphics rendering pipeline, as it relates to programmable shading, the pixel shader and finally Physically-Based Rendering.

4.1 The Graphics Rendering Pipeline

In this section I will present an abstraction of the graphics rendering pipeline and step through some of its stages. (Akenine-Möller, et al., 2008, p. 11).

I will only partially cover this area in the interest of brevity. The understanding of the entire pipeline is useful and I recommend reading the more in-depth explanations available in the reference material, if you are interested.

4.1.1 Architecture

The pipeline can be abstracted into three stages, each feeding data in to the next. The application stage is the program, for example; the game engine. The player might feed the game with keyboard input, changing the position of the camera, moving it forward. (Akenine-Möller, et al., 2008, p. 12)

As said in Real-Time Rendering, “For each frame to be rendered, the application stage feeds the camera position, lighting, and primitives of the model to the next major stage in the pipeline—the geometry stage.” (Akenine-Möller, et al., 2008, p. 25)

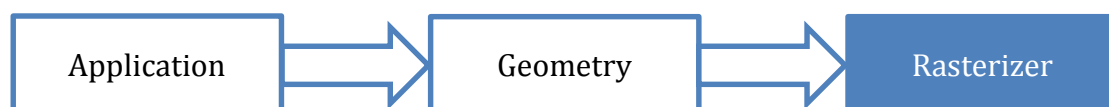


Fig 1. The rendering pipeline, as illustrated by three stages.

² BRDF is the common acronym for the *Bi-directional Reflectance Distribution Function*. “Defines how light is reflected at an opaque surface.” (Wikipedia, n.d.)

4.1.2 The Geometry Stage

In the geometry stage, the primitives (vertices and normals) that are visible to the camera are mapped to the window on the screen, while those that are not are discarded. The Geometry stage feeds the resulting primitives to the final stage, the rasterizer. For our purposes, this is the stage we're the most interested in. (Akenine-Möller, et al., 2008, p. 15)

4.1.3 The Vertex Shader

The vertex shader is a sub-stage of *the geometry stage*. As is explained in Real-Time Rendering, shading involves "computing a shading equation at various points on the objects". The vertex shader computes the calculations once for each vertex of the object. The vertex can store its own position, color and other information. The Result of the vertex shader can for example be the color and the texture coordinates among other things. (Akenine-Möller, et al., 2008, p. 17)

4.1.4 The Rasterizer

The rasterizer receive primitives and converts them into pixels in the window. This process is called rasterization. Remember that each one of the three main stages have each, many more sub-stages. Below are all the sub-stages belonging to the rasterizer. The pixel shading stage is what we're going to look at next. (Akenine-Möller, et al., 2008, p. 21)



Fig 2. The Rasterizer broken down into four sub-stages.

4.1.5 Programmable Shading

There are three stages which are fully programmable; the vertex shader, the geometry shader and the pixel shader. A shader is a computer program that calculates rendering effects (Wikipedia, n.d.), such as how light interacts with a surface.

"While vertex, geometry, and pixel shader programs are necessary to control these stages, they do not exist in a vacuum. First, an individual shader program is not particularly useful in isolation: A vertex shader program feeds its results to a pixel shader. Both programs must be loaded for any work to be done. The programmer must perform some matching of the outputs of the vertex shader to the inputs of the pixel shader." (Akenine-Möller, et al., 2008, p. 45)

4.1.6 The Pixel Shader

More complex shading can be calculated in the pixel shader, "using the interpolated shading data" as input from the vertex shader and geometry stage. Instead of computing the shading per-vertex the pixel shader does it once for each and every pixel of the object. (Hence why it's called per-pixel shading). This stage can employ an important technique called *texturing*, where the shader

'glues' a bitmap image onto the surface of the object. (Akenine-Möller, et al., 2008, p. 43)

4.1.7 Languages

To program these so called shaders, different shading languages have been developed. The most common languages are HLSL (Direct X), CGFX (Direct X & OpenGL), and GLSL (OpenGL). (Wikipedia, n.d.)

4.2 The Shading Model

The shading model can be described as the convergence of several different equations and calculations, all designed to generate different effects of lighting interacting with the surface of an object. Each one of these equations are sometimes referred to as a shading *term*. A complete shading model is generally the result of combining multiple shading terms to generate more realistic visual effects. (Wikipedia, n.d.)

4.2.1 Material Model

The material model is the inputs of the shading model. Some common inputs are bitmap images, which are also known as textures. Sometimes an input is a color (R, G, B) or a single scalar value. The inputs are often exposed in a user-interface, and so can be easily changed and edited to achieve different looks for each material. (Epic Games, n.d., p. 9)

4.2.2 BRDF Terms

Real-Time Rendering describes the BRDF (*Bi-directional Reflectance Distribution Function*) as an abstraction of how light behaves when it interacts with a surface: Such as reflection or sub-surface scattering. (Akenine-Möller, et al., 2008, p. 223)

"In radiometry, the function that is used to describe how a surface reflects light is called the bidirectional reflectance distribution function (BRDF) [932]. As its name implies, it is a function that describes how light is reflected from a surface given two directions—the incoming light direction l and outgoing view direction v ." (Akenine-Möller, et al., 2008, p. 223)

4.2.3 Diffuse Term

The diffuse term models body reflectance, in which case the incoming light of a ray is scattered in every direction. The Lambertian model produces a perfectly *diffuse* surface. The input for the diffuse term is usually only a single color texture. (Wikipedia, n.d.)

4.2.4 Specular Term

The specular term models surface reflectance, in which case all the incoming light of a ray is *reflected* in a single direction. A case of perfect specular reflection would be a mirror, for example. The specular term have two texture inputs. One that controls specular strength and the other for the specular exponent (also known as glossiness). (Wikipedia, n.d.)

4.2.5 Fresnel Term

The Fresnel term is an equation used for angle-dependent reflectance. In laymen's terms it models how reflective a surface is when seen at grazing angles. The Fresnel term have an input for the exponent and strength as well. (Akenine-Möller, et al., 2008, p. 230)

4.2.6 Ambient Term

The ambient term represents incoming light from all directions. In real-time shaders this is simply modelled as a single additive color. The input for the ambient term is then only a single color (R, G, B).

4.2.7 Phong Reflectance

A diffuse term or specular term is not terribly useful in isolation. For the common Phong reflectance the diffuse, specular and ambient term is simply additively combined to form the final result, this is then multiplied with material color and light intensity. The Phong is only partly illustrated below in fig 3. (Wikipedia, n.d.)

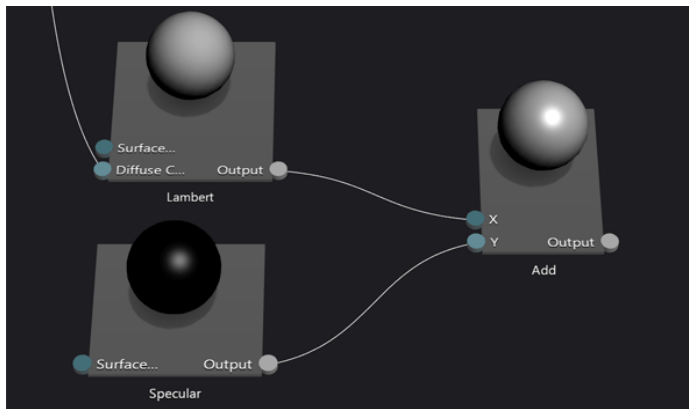


Fig 3. A diffuse term and specular term being added together as visualized in Visual Studio 2013.

4.3 Physically Based Rendering

There has been a move to *PBR* over the last few years. The evidence is the large commercial game engines has adopted it. The latest engine from Epic Games, the Unreal Engine 4, was rolled out the very same month I started writing this paper, featuring *Physically Based Rendering* (April, 2014).

4.3.1 Goals of PBR

To understand the purpose of *PBR* we might look at what goals have been stated about it. Zap Andersson listed in a publication of his (Andersson, 2013, p. 15) a few of the goals he had in mind when creating the famous mental ray shader.

Mia_material, in Anderssons words "*It's a physically plausible mental ray shader.*"

- Conceptual Layer
- Energy Conservation
- "Reflections" & "Specular" are linked
- Fresnel Everywhere (Angle-Dependent Reflectivity)
- Easy Parameters
- Metal Mode
- (Reasonably) Accurate

- Performance

The point of PBR is to make is easier to use and more physically plausible, this is why Anderssons list also match up with real-time rendering in games.

4.3.2 Industry adoption

There were a number of presentations at SIGGRAPH 2013 held by game developers, talking about their shift to be more physically based.

In 2013, Karis presented the goals they (*Epic Games*) had in mind when they “*decided to invest some time in improving our shading model and embrace a more physically based material workflow*”. (Epic Games, n.d., p. 1) Karis continued to state that they wanted to reduce the complexity, “*There should be as few parameters as possible*” and they wanted the values to be simple to understand instead of physical ones such as index of refraction. The material model should be “*Easy to master*” and it should be difficult to create “*physically implausible materials*”.

Other game developers also presented their techniques and implementations of PBR. *Ready at Dawn* (Neubelt & Pettineo, 2013, p. 20) presented what they called *Inheritance-based Materials*, and the three pillars (as they put it) of their material pipeline included “*Compositing*” and “*Run-time layering*” and an “*Inheritance-based data format*”. They go on illustrating their texturing workflow and it is based on layering several pre-made materials in a process where the artist paints masks (as opposed to a color texture).

At Unite 2013, Rod Lopez talked about their plans to implement PBR into the coming version of their game engine known as Unity. He stated that “*today there’s a lot of hacking shaders together...*” and he went on to say that “*it should be easier to do, the artist should have less thinking time...*”. (Lopez, n.d.)

A few game developers talked about how they implemented PBR in games including *Call of Duty: Black Ops II* (Lazarov, 2013) and the upcoming game *The Order: 1886*. (Neubelt & Pettineo, 2013)

4.3.3 Measured Materials

“In most real-time applications, BRDFs are manually selected and their parameters set to achieve a desired look. However, sometimes BRDFs are acquired, or measured, directly from the actual surface.” (Akenine-Möller, et al., 2008)



Fig 4. Measured Base Color Values

A promise of the *Physically-Based Shader* is that it’s based on the real world. With the advent of *PBR* we see shaders are tuned to work with real world scanned data. Measured BRDF models are generally not used in games (at the time of writing this). However measured data can still be turned into useful data, such guideline (R, G, B) colors or just values, even in the form of textures. In researching this I found that there are even coming services aimed at providing

this kind of data. (Quixel, n.d.) But there are also already published data that is useable. (Epic Games, n.d.) (Wilson, n.d.) There are also some tools to convert measured BRDFs into useable values (Legarde, 2012).

4.3.4 The Physically-Based Shading Model

PBR introduces some new principles into the shading model, or more accurately the BRDF model. Listed below is a few common features of Physically-Based shading. The purpose of these features are to make the shading model more physically plausible.

4.3.4.1 Energy Conservation

The surface cannot reflect more than the incoming light. This is not true for a simple phong for example since the specular term is only additive. (Russel, n.d.)

4.3.4.2 Image-Based Lighting

Image-based lighting (IBL) and reflection is not exclusive to PBR, but it is especially important. Environment maps are used, commonly cube maps. Instead of an ambient term that uses a single color, IBL emulates the bounced light from the environment. (Russel, n.d.)

4.3.4.3 Specular and Reflections

In PBR the environment mapped reflections are linked to the specular term. Which is important for energy conservation. Roughness is also linked to (environment map) reflection blurring. Fresnel is also linked to the specular term and is *'always-on'* in every material. (Russel, n.d.)

4.3.5 Unreal Engine 4 Material Model

"Our material model is a simplification of Disney's, with an eye towards efficiency for real-time rendering." (Karis, 2013)

The material model is defined by the textures the artist plugs into the shader. In 2013, Karis presented the new base material model that's used in Unreal Engine 4 in a presentation at SIGGRAPH (p. 9).

Unreal Engine 4's material model, as presented by Karis:

<i>BaseColor</i>	Single color. Easier concept to understand.
<i>Metallic</i>	No need to understand dielectric and conductor reflectance, so less room for error.
<i>Roughness</i>	Very clear in its meaning, whereas <i>gloss</i> needs explaining.
<i>Cavity</i>	Used for small-scale shadowing."

4.3.5.1 Base Color

Base Color is a solid (R, G, B) color. It essentially replaces the diffuse map. The difference is that the *base color* shouldn't contain any shading information (small scale shadowing).

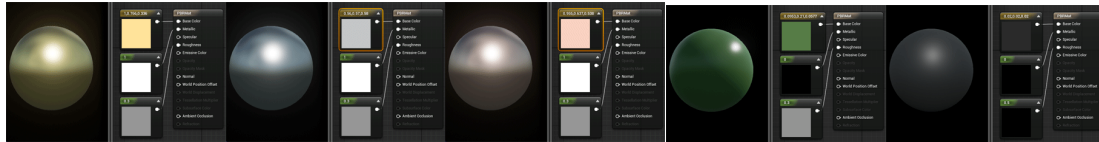


Fig 5. Metal and non-metal materials illustrating different Base Colors.

4.3.5.2 Roughness

Roughness models how rough or smooth a surface is, similar to what glossiness did previously. It is just a scalar value and it can be mapped to a grayscale texture. Roughness is linked to the environment reflection and specular term. A rough surface has blurred soft reflections whereas a smooth surface (low roughness value) have sharp almost mirror-like reflections.

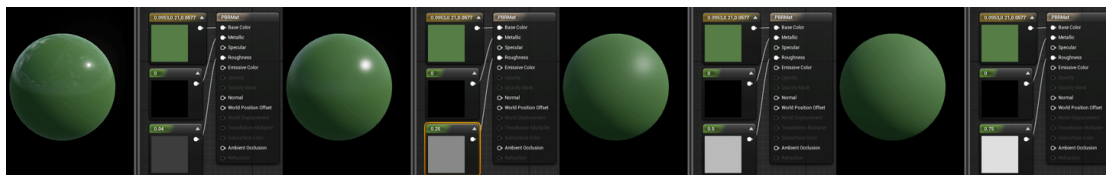


Fig 6. Increasing roughness from left to right.

4.3.5.3 Metallic

Metallic can be treated essentially as a black and white mask, the white areas are metal and the black are not. Any detail that previously went into the specular map should now go into the roughness texture. Metallic can be greyscale values, which can be useful, for example rusted metal or composite materials.

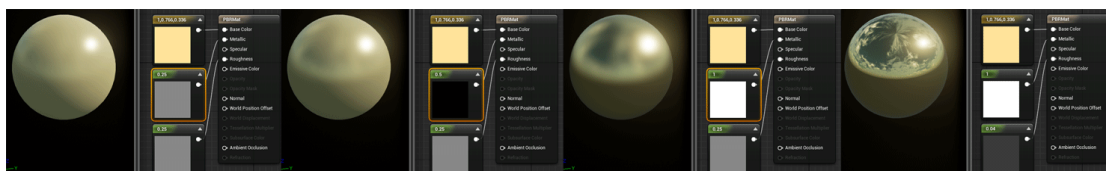


Fig 7. Increasing Metallic value from left to right.

4.3.5.4 Cavity & AO

This map is used for small scale shadowing. Previously this would be baked into the base color to form what is called a diffuse map. Separating the cavity makes the material model in theory more flexible in different lighting conditions.

5 Empirical Research

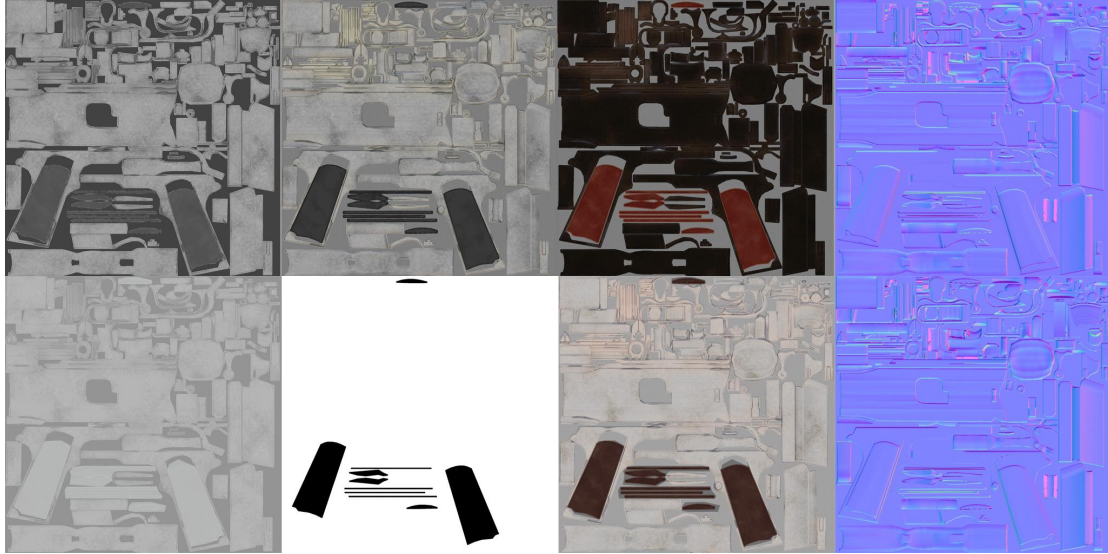


Fig 8. Two texture-sets, each for a different material model.

Above: Glossiness, Specular, Diffuse, Normals.

Below: (Physically-Based) Roughness, Metallic, Base Color, Normals.

In this section I will present the applied research, testing and implementation I did during this project.

5.1 Asset Creation

Physically-based rendering doesn't affect the workflow of 3D modelling or UV-Unwrapping in any remarkable way, I therefore only talk about texturing in this section.

5.2 Texturing for Unreal Engine 4

Creating the textures for a PBR pipeline is different, here I will present the steps I took when creating textures for Unreal Engine 4. I have two materials I will texture, a metal and a non-metal surface, iron and plastic.

5.2.1 Procedural Texturing with DDO

I used Adobe Photoshop, and a plugin called DDO by Quixel. With DDO I could procedurally create the textures faster than if I had done it manually.

5.2.2 Iron


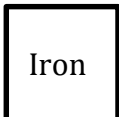

	<i>BaseColor</i> (R, G, B) (0.56, 0.57, 0.58)		<i>Metallic</i> (R, G, B) (1, 1, 1)		<i>Roughness</i> (R, G, B) (0.68, 0.68, 0.68)
---	---	---	---	--	---

Fig 9. The material values I used to texture the iron.

5.2.2.1 Iron Base Color

I arbitrarily decided that an iron surface was what I was looking for. The Unreal Engine documentation has listed a few material values, so I found Iron in that list and used this as starting point. I added textural variation, but kept it always close to the sampled value of iron.

5.2.2.2 Iron Roughness

For the roughness map of the iron I didn't find any published values to start working of. I started by tuning the roughness to visually match it to a photo reference. After the fact, I did find some sources of published roughness values by Sébastien Legarde (2012). In the end I decided to use the value I had come up with by observation, but only for expediency.

5.2.3 Plastic

	<i>BaseColor</i> (R, G, B) (0.22, 0.29, 0.34)		<i>Metallic</i> (R, G, B) (0, 0, 0)		<i>Roughness</i> (R, G, B) (0.7, 0.7, 0.7)
---	---	---	---	--	--

Fig 10. The material values I used texture the plastic.

5.2.3.1 Plastic Base Color

For the *Base Color* of the plastic I actually found a scanned value published by Quixel Megascan (Wilson, n.d.). However, I couldn't use this value straight up as I wanted a different hue, but I kept the value in the same range.

5.2.3.2 Plastic Roughness

Also published at the Marmoset website was a *Roughness* value for plastic (Wilson, n.d.). And again for this I tweaked it to visually match a photo reference.

5.2.3.3 Metallic

The 3D model I created in this example only had two materials and they were clearly separated by geometry. The parts of the texture that is metal I turned completely white. For all of the texture that is supposed to be plastic I set it completely black.



Fig 11. The Final Result displayed with different lighting conditions, in Unreal Engine 4

5.3 BRDF Implementation

In this part of my empirical research I have implemented the common and established BRDF models. Creating shaders is an increasingly simplified process with the advent of node-based visual shader editors. To create the BRDF terms I used the visual shader editor available in Visual Studio 2013.

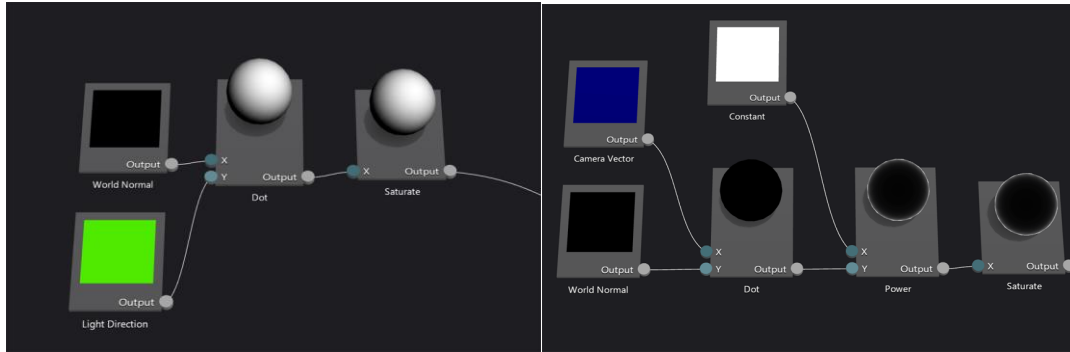


Fig 12. Left: The Diffuse Term, Right: The Fresnel Term (Visual Studio 2013)

I created the three what I consider the most important BRDF terms. The diffuse term, the specular term and the Fresnel term. The diffuse term is a common Lambert. I excluded the example of the ambient term, as it is not very interesting (a single additive color).

5.3.1 Lambert BRDF

The bread and butter BRDF, this is. The calculations are simple and efficient and the inputs are few. Lambert reflectance is calculated by taking the dot product of the surface's *normal vector*, N and the *light-direction vector*, L . The color of the surface C and the light intensity I_L is then multiplied onto this. It is also worth noting that N and L is *normalized*.

$$I_D = L \cdot NCI_L$$

I_D is the resulting reflecting light intensity. (Wikipedia, n.d.) Fig 12. Illustrates my implementation of the diffuse term in visual studio 2013.

5.3.2 Fresnel BRDF

The Fresnel BRDF term is calculated by the exponentiation of the dot product of the normalized camera vector V and surface normal vector L . n is the exponent.

$$I_D = (L \cdot V)^n$$

It's worth noting that this is probably a gross simplification, but it's my understanding of the calculations at the moment of writing this paper. However I would recommend you use other sources for your own implementations. Fig 12, illustrates my implementation of Fresnel in Visual Studio 2013.

5.3.3 Phong Specular BRDF

The specular BRDF is arguably the most interesting but also most complex. I cannot do the calculations justice as I do not understand them intrinsically enough, so I will not attempt it for this paper. However Fig 13, Shows my implementation of the specular term in Visual Studio 2013.

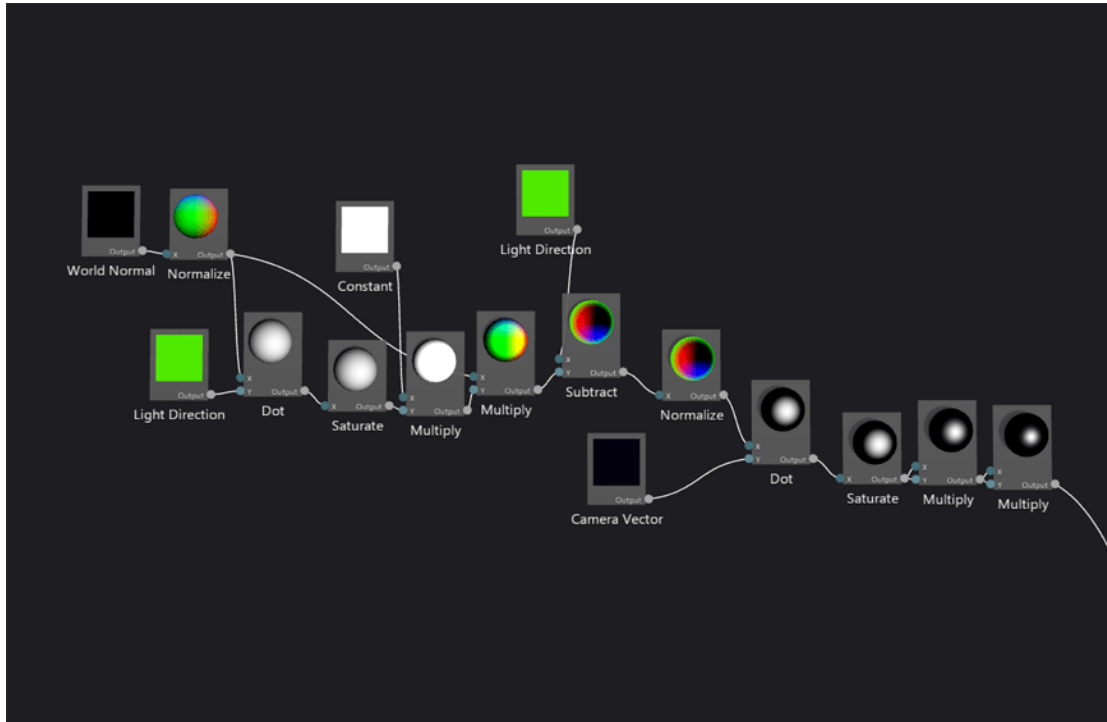


Fig 13. The Specular Term, as constructed in Visual Studio 2013

6 Results

PBR is impactful in many ways, I think that a pipeline with PBR principles is a step in the right direction. It is possible to use scanned data when texturing – this helps with materials definition or recreating real world materials.

- It is a more standardized pipeline then that of the last generation.
- More efficient, less trial and error.
- Can use scanned real world material values or textures.
- Easier to make materials look good in all lighting conditions.

6.1 A new graphics pipeline

There is a new material model, requiring new parameters. When I got used to the new material model, it was way easier to create good looking and physically plausible materials. *Before* we had to “bake” lighting information into the textures. The parameters were also comparatively obtuse. Now with PBR, the material model feels much more natural, this is the way it should have been all along.

6.2 Industry Adoption

It seems that a lot of the large game studios and engines are using it. The fact that Unreal Engine 4 have it implemented and soon Unity, is two strong indicators of the industry adoption.

7 Discussion

In this report I decided to only investigate one implementation of physically based rendering. However it would have been very valuable to compare a few differing implementations. One problem of changing something as the shading model is that it requires re-learning of techniques by the staff.

Varying implementations can also make it confusing at times, the problem is that resources from can't be shared.

7.1 The shading model

The new shading model is quite different than the old. The job of the shading model is to emulate how light interacts with the surface of an object. In computer graphics this task is broken down into the component parts of the shading model, these are called the shading terms or BRDF terms. In computer graphics there are generally only a few different interactions that happens when a light ray bounces of a surface such as reflection, sub-surface scattering or refraction. In the real world this is all governed by physics, but in real-time graphics we have to design calculations that visually match what we see in the real world. The limitations of real-time graphics is that these calculations needs to perform very efficiently, many times per second. This limitation requires us to simplify the shading calculations, and so the calculations are not actually physically accurate but merely an approximation.

Physically-Based Rendering effect the shading calculations, with the expressed intention of making them more physically accurate. Although there are several varying implementations (of PBR) I found that there are a few common denominators. One of the new additions to the shading model is called 'Energy Conservation'. When a ray of light reflects of a surface, the maximum amount of light reflected cannot be exceed the amount of the incoming light ray. This is what's called energy conservation. This not only leads to the shading model being more physically accurate, I find that it improves the pipeline and makes it more efficient for the artist. A shading model that's not energy conservative adds more responsibility to the artist. The artist have to manually tune the material values to make it physically plausible while texturing. With an energy conservative shading model this is automatically handled by the calculations instead of the artist.

7.2 The material model

The shading model is the input parameters of the shading model. With this in mind, the new shading model bring with it a new material model. As with the shading model, there are multiple varying implementations and I've looked at the one by Epic Games implemented in their Unreal Engine 4. Previously there were parameter-inputs for specular intensity/color and the glossiness, in addition to a diffuse map. The new shading model "replaces" the specular intensity as this is now directly dependent on the specular exponent, Epic calls this new parameter *Roughness*. The benefit of the Roughness parameter is that it introduces a link, or a relationship between the specular intensity and the glossiness. The Roughness parameter also controls the blurriness/sharpness of the image-based reflection. The result of the relationship is that the material model is more physically accurate. One of the most important use-cases of specular color was being able to define metallic materials. Epic introduced a

parameter into the new material model for this purpose called *Metallic*. They also “split” the diffuse map into two parameters, *BaseColor* and *Cavity*. The diffuse map contains both color and shadow information baked together. The new model separate the color information into the *BaseColor* parameter and the shadow information into the *Cavity* parameter.

7.3 The texturing pipeline

The new material model does have consequences to the texturing pipeline. The Roughness and Metallic maps replace the functionality of the specular color and glossiness maps. I find that the physically-based model is easier to work with. With a *specular color* model, the artist have to control both the specular intensity and color in one map. In the old model, *specular color* defines if your material is a *metal* or *non-metal*, however editing the *specular color* is a slow task in my experience. The artist also have to manually tweak and balance the specular intensity and glossiness parameter, so that the material looks good and physically plausible. The physically-based model removes a lot of this manual labor. The artist only have to control two parameters, roughness and metallic. The benefit is that both roughness and metallic can respectively be authored as a greyscale map. The specular intensity is no longer needed to be manually edited, as this is linked and therefore controlled by the roughness parameter. The advantage is quantifiable, there is one less parameter the artist has to edit and this can be a huge time-saver. Previously different surfaces such as wood or plastic or metal needed a different specular color to be correctly defined. Instead of having to edit the specular color, we use the Metallic map to define metal and non-metal surfaces. This means that all non-metals are always defined the same way with a metallic value of zero (black) while all metals are defined with a value of 1 (white). This is in my experience much easier to work with and potentially another huge time-saver.

The final parameters of the physically-based material model is *BaseColor* and *Cavity*. *BaseColor* is what is used to define the local color of the material. There are a few important differences when creating the *BaseColor* map compared to the diffuse map. For metal materials *BaseColor* is the color that is used to differentiate between different colored metals, say gold or copper. Previously this was accomplished with the specular color parameter. The benefit of this is that the artist only has to set the material color once in the *BaseColor*. All small-scale shadowing that were previously baked into diffuse map is now in a separate map called *Cavity*. This means that *BaseColor* shouldn't have any shadow information. One problem with this is that if you're using photos as textures you have to remove any shadowing. The benefit is that you can quickly rebake the cavity map. Previously you would have to open the diffuse and add the cavity map in a layer and set it to multiply.

Finally, the new physically-based pipeline allows the use of scan-data like never before. The advantage of this is that the artist can look up colors and values for real world materials. This makes it faster to recreate the materials without a trial and error approach.

8 Conclusions

8.1 Physically-Based Shading

Companies such as Epic Games and Unity has talked about the goals of Physically-Based Rendering. It should make shading in games more physically accurate, while making the process of creating graphics and art easier and more efficient during production. The significant, new changes I've observed are in the new shading and material models. Physically-Based Rendering have changed these models in a number of ways. The new parameters in the shading model is designed to be more physically accurate. There are now new "*relationships*" between parameters, whereas previously the artist would have to manually control or edit each value. Another significant improvement is the ability to use scan-data when texturing, taking out some of the guess-work of previous pipelines. The result is a two-fold benefit. Firstly, creating materials are more efficient, in part thanks to a reduced amount of parameters. Secondly, it's easier to achieve good visual quality.

8.2 Future Developments

Unreal Engine 4 has already shipped with Physically-Based Rendering as one of its selling points. Unity has announced that the next version of their engine will also implement it. I expect we will see it being used in more games in the months and years to come.

The fact that we now can use scan-data in the texturing process has created an opportunity for business. Quixel has announced their new service where artists and companies will be able to buy scanned texture for use in a physically-based pipeline. (Quixel, n.d.)

There is also a risk, with new techniques being adopted that we'll see some different implementations of physically based principles. However, I still personally expect a positive change because it seems that the industry is quite open these days and share a lot of information at conferences like SIGGRAPH. A lot of the information I've found is from game developers at SIGGRAPH so I think that's a good sign.

9 References

- Akenine-Möller, T., Haines, E. & Hoffman, N., 2008. *Real-Time Rendering: Third Edition*. s.l.:Taylor & Francis Group.
- Andersson, Z., 2013. *Everything You Always Wanted to Know About mia_material*(*But Were Afraid to Ask)*. [Online]
Available at: http://blog.selfshadow.com/publications/s2013-shading-course/andersson/s2013_pbs_mia_notes.pdf
[Accessed 22 May 2014].
- Epic Games, n.d. *Unreal Engine 4 Documentation: Physically Based Materials*. [Online]
Available at: <https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/index.html>
[Accessed 22 May 2014].
- Hansson, N., 2013. *Feeding A Physically Based Lighting Model*. [Online]
Available at: <http://www.defrostgames.com/2013/05/>
[Accessed 22 May 2014].
- Hoffman, N., 2013. *Background: Physics and Math of Shading*. s.l., s.n.
- Karis, B., 2013. *Real Shading in Unreal Engine 4*. [Online]
Available at: http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf
[Accessed 22 May 2014].
- Lazarov, D., 2013. *Getting More Physical in Call of Duty Black Ops II*. [Online]
Available at: http://blog.selfshadow.com/publications/s2013-shading-course/lazarov/s2013_pbs_black_ops_2_slides_v2.pdf
[Accessed 22 May 2014].
- Legarde, S., 2012. *DONTNOD specular and glossiness chart*. [Online]
Available at: <http://seblagarde.wordpress.com/2012/04/30/dontnod-specular-and-glossiness-chart/>
[Accessed 24 May 2014].
- Lopez, R., n.d. *Physically Based Shading Upcoming in Unity 5-0*. [Online]
Available at: <http://blogs.unity3d.com/2014/03/20/physically-based-shading-upcoming-in-unity-5-0/>
[Accessed 24 June 2014].
- Neubelt, D. & Pettineo, M., 2013. *Crafting a Next-Gen Material Pipeline for The Order: 1886*. [Online]
Available at: http://blog.selfshadow.com/publications/s2013-shading-course/rad/s2013_pbs_rad_slides.pdf
[Accessed 22 May 2014].
- Quixel, n.d. *Megascans*. [Online]
Available at: <http://dev.quixel.se/megascans>
[Accessed 25 May 2014].
- Quixel, n.d. *MegaScans*. [Online]
Available at: <http://dev.quixel.se/megascans>
[Accessed 24 June 2014].
- Russel, J., n.d. *Basic Theory of Physically Based Rendering*. [Online]
Available at: <http://www.marmoset.co/toolbag/learn/pbr-theory>
[Accessed 24 June 2014].

Wikipedia, n.d. *Bidirectional Reflectance Distribution Function*. [Online]
Available at: http://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function
[Accessed 12 June 2014].

Wikipedia, n.d. *Diffuse Reflection*. [Online]
Available at: http://en.wikipedia.org/wiki/Diffuse_reflection
[Accessed 25 May 2014].

Wikipedia, n.d. *Lambertian*. [Online]
Available at: <http://en.wikipedia.org/wiki/Lambertian>
[Accessed 25 May 2014].

Wikipedia, n.d. *Phong Shading*. [Online]
Available at: http://en.wikipedia.org/wiki/Phong_shading
[Accessed 25 May 2014].

Wikipedia, n.d. *Shader*. [Online]
Available at: <http://en.wikipedia.org/wiki/Shader>
[Accessed 25 May 2014].

Wikipedia, n.d. *Shading*. [Online]
Available at: <http://en.wikipedia.org/wiki/Shading>
[Accessed 12 June 2014].

Wikipedia, n.d. *Shading Language*. [Online]
Available at: http://en.wikipedia.org/wiki/Shading_language
[Accessed 25 May 2014].

Wikipedia, n.d. *Specular Reflection*. [Online]
Available at: http://en.wikipedia.org/wiki/Specular_reflection
[Accessed 25 May 2014].

Wilson, J., n.d. *Physically Based Rendering, And You Can Too!*. [Online]
Available at: <http://www.marmoset.co/toolbag/learn/pbr-practice>
[Accessed 24 May 2014].