Escola das Artes da Universidade Católica Portuguesa

Master's Degree in Sound and Image

Production of 3D Animated Short Films in Unity 5: Can Game Engines Replace the Traditional Methods?

Specialization in Computer Animation – 2016/17

Luís António Gomes Tarrafa Ramos

Supervisor: Doutora Sahra Kunz

Co-Supervisor: Mestre André Cruz

April 2017

Dedication

I dedicate this work to my parents António Ramos and Cristina Ramos, who trusted me by giving me the opportunity to continue studying and without whom none of my academic progress would have been possible, and to Ânia Guerra for all the support she gave me and for always trying to cheer me up when I needed it.

Acknowledgements

I would like to thank Sahra Kunz and André Cruz, this work’s supervisor and co-supervisor respectively, for all the support and orientation provided during the development of this dissertation. I also acknowledge all the elements of Universidade Católica Portuguesa that were directly or indirectly involved in the development of this work or contributed in any way to its success.

I would like to thank everyone that gave me moral support or useful critique and suggestions during the development of this dissertation.

Abstract

In 3D animation cinema, the elements of a scene are created by artists using computer software. To generate the final result, there must be a conversion (rendering) of the three-dimensional models to two-dimensional images (frames) that will later be joined together and edited into a video format.

3D animation films have traditionally been rendered using pre-rendering engines, a time consuming and expensive process that usually requires the use of multiple computers rendering at the same time (render farms), renders which may need to be repeated if the results are not ideal.

Videogames, on the other hand, are reactive applications where the player may have different possible courses of action that will generate distinct results. In those cases, it is necessary that the engine waits for the player's input before it calculates the following frames. To allow for fast calculations in real time, 3D game developers use game engines that incorporate real time rendering methods which can generate images much faster than the pre-rendering engines mentioned above.

To be able to generate a large number of frames per second, there must be an optimization of the entire scene, in order to reduce the number of necessary calculations. That optimization is created by using techniques, practices and tools that are not commonly used by animation cinema professionals.

Due to that optimization necessity, videogames always had a lower graphic quality than that of animated films, where each frame is rendered separately and takes as long as necessary to obtain the required result.

Physically Based Rendering (PBR) technology is one of the methods incorporated by some rendering engines for the generation of physically accurate results, using calculations that follow the laws of physics as it happens in the real world and creating more realistic images which require less effort, not only from the artist but also from the equipment. The incorporation of PBR in game engines allowed for high graphic quality generated results in real time, gradually closing the visual quality gap between videogames and animated cinema.

Recently, game engines such as Unity and Unreal Engine started to be used – mostly by the companies that created the engine, as a proof of concept – for rendering 3D animated films. This could lead to changes in the animation cinema production methods by the studios that, until now, have used traditional pre-rendering methods.

Keywords: 3D Animation, short film, real-time rendering, game engine

Resumo

No cinema de animação 3D, os elementos de uma cena são criados por artistas através da utilização de programas de computador. Para gerar o resultado final, é necessário fazer-se uma conversão (*render*) dos modelos tri-dimensionais para imagens bi-dimensionais (*frames*), que posteriormente serão unidas e editadas para um formato de vídeo.

Tradicionalmente, o *rendering* de filmes de animação 3D é feita através de motores de *pre-rendering*, um processo demorado e dispendioso que geralmente requer a utilização de múltiplos computadores a trabalhar em simultâneo (*render farms*), e que poderá ter que ser repetido caso os resultados obtidos não sejam ideais.

Os videojogos, por outro lado, são aplicações reactivas, onde o jogador pode ter várias sequências de acções, que poderão gerar resultados distintos. Nesses casos, é necessário o motor de jogo esperar pela acção do jogador antes de calcular as imagens seguintes. Para possibilitar cálculos rápidos em tempo-real, os criadores de jogos 3D usam motores de jogo que incorporam métodos de renderização em tempo-real que conseguem gerar imagens muito mais rápido do que os motores de *pre-rendering* mencionados acima.

Para conseguir gerar um grande número de imagens por segundo, é necessário existir uma optimização de toda a cena, para reduzir o número de cálculos necessários. Essa optimização é criada através da utilização de técnicas, práticas e ferramentas que, geralmente, não são utiliadas por profissionais da área de cinema de animação.

Devido a essa necessidade de optimização, os videojogos sempre tiveram uma qualidade gráfica inferior à dos filmes de animação, onde o *render* de cada imagem é gerado separadamente e pode levar tanto tempo quanto for necessário para obter o resultado desejado.

A tecnologia de *Rendering* Baseado em Física (*Physically Based Rendering – PBR*) é um dos métodos incorporados por alguns motores de *rendering* para a geração de resultados físicamente correctos, usando cálculos que seguem as leis da física, tal como acontece no mundo real e criando imagens mais realistas necessitando de menos esforço, não só da parte do artista mas também do equipamento. A incorporação de PBR em motores de jogo possibilitou resultados gerados em tempo-real com grande qualidade gráfica, o que gradualmente vai aproximando a qualidade visual dos videojogos à do cinema de animação.

Recentemente, motores de jogo como o Unity e o Unreal Engine começaram a ser utilizados – maioritariamente pelas companhias que criaram o motor de jogo, como prova de conceito – para renderização de filmes de animação 3D. Este passo poderá levar a mudanças nos métodos de produção do cinema de animação em estúdios que, até agora, utilizaram métodos de pré-renderização tradicionais.

Keywords: Animação 3D, curta-metragem, *rendering* em tempo-real, motor de jogo

Table of Contents

List of Figures ... 1

List of Abbreviations ... 3

1. Introduction ... 5

1.1. Subject of Investigation ... 5

1.2. Methodology ... 5

1.3. Structure of the Dissertation ... 6

2. State of the Art ... 7

2.1. The Rise of Animation Cinema and Technology ... 7

2.1.1. Animated Films ... 8

2.2. Real-Time Rendering Engines ... 9

2.2.1. 3D Game Engines ... 9

2.2.2. Pre-Rendering vs Real-Time Rendering ... 10

2.3. Physically Based Rendering ... 11

2.3.1. Light Mechanics in PBR ... 14

2.4. 3D Animated Film Production Technologies ... 17

2.4.1. Modelling and Sculpting ... 17

2.4.2. Rigging ... 19

2.4.3. Animation ... 22

2.4.4. Textures and Maps ... 23

2.4.5. Lighting and Rendering ... 23

2.4.6. Post-Production and VFX ... 25

2.4.7. Other Technologies and Techniques ... 27

2.5. Real-Time Rendered Short Films ... 27

2.5.1. The Gift (2016) – Marza Animation Planet Inc. ... 27

2.5.2. Adam (2016) – Unity Technologies ... 28

2.6. Chapter Conclusion ... 30

3. Analysis of the Production Pipeline ... 32

3.1. Traditional Pipeline Overview ... 32

3.1.1. Pre-Production ... 32

3.1.1. Production ... 33

3.1.1. Post-Production ... 35

3.2. Planning and Methodology ... 35

3.2.1. Software Set-Up 35
3.2.2. Creating 3D Assets 35
3.2.3. Textures 36
3.2.4. Rendering Paths 36
3.2.5. Lighting and Cameras Set Up 37
3.2.6. Animation Sequencing Method 38
3.2.7. Other Optimization Methods 38
3.3. Modelling 38
3.4. Procedural Asset Creation and Animation 40
3.4.1. Physics and Simulation 41
3.5. Rigging and Animation 42
3.6. Materials and Textures 43
3.6.1. PBR Workflow 43
3.7. Lighting and Light Baking 45
3.7.1. Light Types 45
3.7.2. Global Illumination and Light Baking 45
3.7.3. Light Probes and Reflection Probes 46
3.8. Rendering 47
3.8.1. Single-Pass Rendering 47
3.8.2. Multi-Pass Rendering 47
3.9. Sound and Post-Production 48
3.9.1. Sound 48
3.9.2. Image Effects and Shaders 48
3.9.3. Lookup Textures (LUTs) 49
3.10. Chapter Conclusion 49
4. Final Conclusions and Future Work Perspectives 51
4.1. Comparison Summary 51
4.2. Final Product Analysis 51
4.3. Is Real-Time Rendering a Viable Option? 52
4.4. The Future of Animated Cinema 52
References 53
ANNEX A – 3D Production Pipeline 62
ANNEX B – PBR Workflows 63
ANNEX C – Before and After Light Baking in Unity 64

List of Figures

Fig. 1 Unreal Engine 4 blueprint example, *retrieved from https://docs.unrealengine.com/latest/INT/Engine/Blueprints/ on June 21, 2017. Copyright Epic Games, Inc. (2017)* 10

Fig. 2 Life of Pi (2012) screen capture, before and after VFX, *retrieved from https://digitalsynopsis.com/design/movies-before-after-green-screen-cgi/ on June 21, 2017. Copyright Digital Synopsis (n.d.)* 12

Fig. 3 Raytracing visual representation, *retrieved from https://blog.codinghorror.com/real-time-raytracing/ on June 21, 2017. Copyright Atwood, J. (2008)* 13

Fig. 4 Light interaction, *retrieved from https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/ on June 21, 2017. Copyright Russell, J. (2015)* 15

Fig. 5 Light interacting with microsurface detail, *retrieved from Light and Matter: The Theory of Physically-Based Rendering and Shading, vol. 1 on June 21, 2017. Copyright McDermott, W. (2015)* 16

Fig. 6 Facial edge flow in 3D modelling, *retrieved from http://area.autodesk.jp/column/tutorial/character_arpeggio/01_modeling/ on June 21, 2017. Copyright Akasaki, H. (2016)* 18

Fig. 7 Character rig in Autodesk Maya, *retrieved from https://cgi.tutsplus.com/tutorials/modelling-rigging-a-cartoon-parrot-in-maya-part-9--cms-23933 on June 21, 2017. Copyright Kumari, S. (2015)* 20

Fig. 8 Rig control curves, *retrieved from https://animationblog.wordpress.com/2016/01/28/r-d-blog-week-2-part-2/ on June 21, 2017. Copyright aneesharai22 (2016)*........ 21

Fig. 9 Animation interpolation, *retrieved from Multiresolution Mean Shift Clustering Algorithm for Shape Interpolation on June 21, 2017. Copyright Chu, H.-K., & Lee, T.-Y. (2009)*........ 22

Fig. 10 HDRI example, *retrieved from http://www.lughertexture.com/spherical-hdri-360/spherical-360-nature-trees-plants-sky-hdri on June 21, 2017. Copyright IuGher (2015)* 24

Fig. 11 Render passes, *retrieved from http://marcelmarkov.deviantart.com/art/Zbrush-Bust-Render-Passes-337452354 on June 21, 2017. Copyright MarcelMarkov (2012)* 25

Fig. 12 Marza's tests on the Alembic Importer for Unity, *retrieved from Making of The Gift: Game Changer in Film Production on June 21, 2017. Copyright Unite Europe (2016)*........ 28

Fig. 13 Adam's screenshot inside Unity, *retrieved from ADAM - Assets Creation for the Real Time Short Film on June 21, 2017. Copyright Tamnev, P. (2016)*..... 29

Fig. 14 Storyboard excerpt from Pixar's Brave (2012) feature film, *retrieved from http://pixar-animation.weebly.com/pixars-animation-process.html on June 21, 2017. Copyright Vardanega, J. (2013)* 33

Fig. 15 Character in the default modelling pose, *retrieved from 3D Animation Essentials on June 21, 2017. Copyright The Basement Design + Motion (2012)* 33

Fig. 16 Reflections using LDR and HDR environments, *retrieved from https://gamedev.stackexchange.com/questions/62836/does-hdr-rendering-have-any-benefits-if-bloom-wont-be-applied on June 21, 2017. Copyright Reed, N. (2013)*.. 37

Fig. 17 Different Levels of Detail on the same object, *retrieved from http://evgeny-3d.com/3d-modeling/scull.html on June 21, 2017. Copyright Lyapin, E. (n.d.)* 40

Fig. 18 Unity's Animator Controller, *retrieved from https://blogs.unity3d.com/2016/08/31/adam-animation-for-the-real-time-short-film/ on June 21, 2017. Copyright Nechevski, K. (2016)*......................... 42

Fig. 19 Lookup texture in Unity, *retrieved from https://docs.unity3d.com/550/Documentation/Manual/script-ColorCorrectionLookup.html on June 21, 2017. Copyright Unity Technologies (2017)* 49

List of Abbreviations

AO	Ambient Occlusion
AR	Augmented Reality
BRDF	Bidirectional Reflectance Distribution Function
CG	Computer Graphics
CPU	Central Processing Unity
FK	Forward Kinematics
GI	Global Illumination
GPU	Graphics Processing Unit
HDR	High Dynamic Range
HDRI	High Dynamic Range Image
IK	Inverse Kinematics
LDR	Low Dynamic Range
LOD	Level Of Detail
LUT	Lookup Texture
MoCap	Motion Capture
MSAA	Multi-Sample Anti-Aliasing
PBR	Physically Based Rendering
RGB	Red Green Blue
sRGB	Standard Red Green Blue
Texel	Texture Pixel
Triple-I	Information International Incorporated
UE4	Unreal Engine 4
VR	Virtual Reality

1. Introduction

1.1. Subject of Investigation

The main goal of this dissertation is to establish a comparison between two rendering methods used in 3D animation films production: pre-rendering – the traditional method – and real-time rendering, a technology that only recently started being used for animated short films production, and often just as a proof of concept.

The new methods and technologies involved in real-time rendering for film production have the potential to affect the production methods that have been in use for decades, creating a more efficient workflow and allowing for more artist-friendly approaches to some steps of the production pipeline.

By changing the technologies used in animated film production, 3D animation artists may be required to expand their knowledge of game engines and game development in general, while still being able to use their knowledge of traditional rendering methods.

The combination of distinct technological areas or methods can generate new forms of art, such as the incorporation of user interactivity, Virtual Reality (VR) or Augmented Reality (AR) systems in animated films.

Being a recent technology still under development, that could affect the current production methods and change the future of animated cinema, makes this subject worthy of investigation.

1.2. Methodology

In order to create a comparison between the two methods mentioned above, every step of the production pipeline of each method must be analysed and compared, and all the advantages and disadvantages taken into account.

By the end of the research, using the combined collected data from each step, we will be able to conclude in which situations the use of each process will be more advantageous, and possibly speculate if, in the future, there will be room for both methods in large studios, either by using only one of them or by combining the strengths of both.

The research process will follow an explorative and comparative methodology, where the first approach will be a general investigation of existent sources of material and projects developed or in development that relate to the issue being studied. The initial material will be composed of text documents (papers, articles, books, etc.), instructional videos, interviews and/or public talks, and visualization of animated 3D films created using both processes.

This information will compose the first draft of the dissertation, and its analysis will allow for a better planning of the following steps, by showing where there is a lack of information or a need for practical examples.

After these steps are concluded, a more in-depth approach will be taken in the investigation of specific techniques or technologies used in each method, detailing the production pipelines commonly used or analysing other methods that could accomplish a specific goal or result.

The final data will be condensed and combined, and then be put through a critical analysis process in order to reach relevant conclusions.

1.3. Structure of the Dissertation

This dissertation will be divided into three main chapters.

The State of the Art (chapter 2) will be focused on exposing and explaining the existing technology and techniques used in traditional and real-time rendering. This chapter will also introduce the notions and concepts required to understand the following chapters.

The Analysis of the Production Pipeline (chapter 3) will contain an analysis of the production steps used in traditional rendering and use that analysis as a base to compare it with each step to be used in a real-time rendering pipeline.

In Final Conclusions and Future Work Perspectives (chapter 4), the data collected in chapter 3 and the conclusions taken from it will be reviewed and condensed into a critical analysis to reach conclusions regarding the advantages, disadvantages and the future of real-time rendering in 3D animated short films production.

2. State of the Art

2.1. The Rise of Animation Cinema and Technology

There are two primary industries using 3D animation today: the entertainment industry, which is the most recognized and includes film, television, video games and advertising, and the scientific industry, which includes medicine, law, architecture and product visualization. 3D animation is a very recent technology, and new ways of using it are constantly appearing, such as augmented reality, using 3D to create art such as sculptures, and projection mapping (Beane, 2012, pp. 2-10).

Despite that most of the technologies being used today by the 3D animated film industry have been developed before the 21st century, from 3D animation software and ray-tracing rendering technology to real-time rendering engines[1]. Nowadays, the majority of the developments in 3D animation technology are improvements of already existing processes in terms of efficiency and capabilities, along with the development of the hardware used in the production of 3D animation. These developments allow the artists to create more realistic and complex models, simulations and animations with less effort.

The terms Computer Graphics (CG) and Computer Animation were created in the 1960s. The term *Computer Graphics* is credited to William Fetter in 1960. 3D computer animation and rendering, and many of the techniques we still use today were only invented in the 1970s. In this decade, the first 3D animation studios were created, such as Information International Incorporated (Triple-I), Robert Abel and Associates, Digital Effects, and Lucasfilm which also had a CG division called Graphics Group which would eventually become Pixar (Beane, 2012, pp. 10-19).

"This decade [1960s] is when we saw the computer evolve from a strictly calculating device into a tool that allowed for creation and change." (Beane, 2012, p. 11)

The first commercially available 3D animation software was created by Wavefront Technologies in 1984. Before that, companies that created 3D animation needed to write their own proprietary software. It was only in the 1990s that 3D graphics began to be used in video game consoles, with the release of the Sony Playstation and Nintendo 64 systems (Beane, 2012, pp. 10-19), with one of the most relevant games being *Legend of Zelda: Ocarina of Time* (1998, Nintendo 64) (Kerlow, 2004, p. 28).

In the late 1990s and beginning of 2000s, films began to have more complex digital effects. *Terminator II* (1991) had approximately 150 visual effects shots, *Batman Forever* (1995) had around 250, *Titanic* (1997) had close to 550, *Armageddon* (1998) had around 240, *Godzilla* (1998) had close to 400 and *How the Grinch Stole Christmas* (2000) had around 600 (Kerlow, 2004, p. 27).

Since then, 3D animation technology in films and advertising has become so realistic that most people cannot notice they are looking at a CG object or effect (Beane, 2012, pp. 10-19).

[1] Vd. chapters 2.2 – Real-Time Rendering Engines and 2.3 – Physically Based Rendering and respective subchapters for a more detailed analysis on each technology

2.1.1. Animated Films

Pixar is one of the most influential animated film studios in the history of 3D animated cinema. Pixar's *Toy Story* (1995) was the first animated feature film to be entirely created with three-dimensional computer animation techniques (Kerlow, 2004, p. 26), and it was a huge success. It was expected to get a return of $100-200 million, and although it had only a budget of $30 million and only 110 staff members, it had a worldwide gross of $362 million. The technology was not yet fully developed, but the captivating story and characters made up for it. The film took around 800 thousand machine hours to render – 2 to 15 hours per frame (Bettinger, 2017, paras. 3-15).

George Lucas and Industrial Light & Magic had already revolutionized the film industry with their first *Star Wars* trilogy (1977-83), and they did it again in the prequels trilogy (1999-2005) by using digital effects and computer animated 3D characters and environments in a live-action film (Lobo, 2014, paas. 5-8). In *Star Wars: Episode II* (2002) there were 2200 visual effects shots, with 10200 visual effects elements, 5 million frames, 929 animated shots, and a crew of over 250 digital artists (Kerlow, 2004, p. 30).

The prequels were not well received by the fans due to the excess of computer animated characters, although just 2 years after the first movie of the prequel, *Lord of the Rings* (2001) introduced a very well received Gollum character which was almost entirely created with CGI (Lobo, 2014, para. 6), while also accomplishing great results with crowd simulation (Beane, 2012, pp. 10-19). Despite the unwillingness of many cinema lovers to accept that visual effects and CGI are the future of cinema, the technological developments in these areas have reached the point where most people cannot make out if a visual element is a practical or a visual effect. However, although the quality of visual effects has improved immensely in just a few years, there is still a long way to go until we can create believable humanoid CGI characters (Lobo, 2014, paras. 18-19).

Monsters Inc. (2001) was a very successful film, with a worldwide gross of $525 million. It had a lot of technological improvements relative to Pixar's first film, particularly in the physics department (Beane, 2012, pp. 10-19). As stated by the Collider entertainment news website writer Brendan Bettinger (2017, para. 47), there were "2.3 million individually animated hairs on Sulley. It took 11-12 hours to render a single frame with Sulley in it."

Finding Nemo (2003) took Pixar to another level, with a worldwide gross of $868 million. It was on the top 10 of both domestic and worldwide box office charts, and it broke the first-day record for home-release DVD sales with 8 million copies. By the end of 2006 it had sold 40 million copies (Bettinger, 2017, paras. 51-60).

Up (2009) was another important mark in Pixar's history, grossing $731 million worldwide. With *Up*, Pixar started creating 3D animated films that used the 3D screens technology in the cinemas, and even made re-releases of early films with that technology. It was also the first Pixar movie and the first animated movie since Disney's *The Beauty and the Beast* to be nominated for Best Picture at the Academy Awards (Bettinger, 2017, paras. 108-119).

Toy Story 3 (2010) grossed $1.1 billion, almost three times more than *Toy Story* (1995). All the models needed to be remade from scratch for this film, because the old character files could not be opened in the new software (Bettinger, 2017, paras. 120-131).

The technology in *Monsters University* (2013) surpassed previous Pixar movies. There were 400 different monsters in the movie, with the average shot containing more than 25. Sully's character had 5.5 million individual hairs on his body, double of what it had in *Monsters, Inc.* (2001), and 100 million CPU (Central Processing Unit) hours[2] were needed to render the final edit – the film took 2 years to be rendered, using 2.000 computers (Bettinger, 2017, paras. 155-169).

2.2. Real-Time Rendering Engines

2.2.1. 3D Game Engines

"The great popularity and growth experienced by computer games and platform games translated into many jobs for three-dimensional computer animators (...)" (Kerlow, 2004, p. 13)

The most popular 3D Game Engines available to the public today are Unity 5, Unreal Engine 4 (UE4) and CryENGINE. All of them have great graphical capabilities with technologies such as Physically Based Shading and real-time Global Illumination, and include particle systems, physics engines, advanced animation systems, and accept major 3D applications' file formats.

"Global illumination, or 'GI', is a term used to describe a range of techniques and mathematical models which attempt to simulate the complex behaviour of light as it bounces and interacts with the world." (Unity Technologies, Introduction to Lighting and Rendering, 2017, para. 1)

All the engines are able to produce somewhat similar results, depending more on the skill of the developers than the capabilities of the engine itself. When choosing an engine, unless developing for a specific platform that only one of them supports, developers should experiment with all of them and choose the tool that suits them more.

According to Mark Masters (2014, para. 20), instructor at Pluralsight/Digital Tutors, CryENGINE has the steeper learning curve of the three, with the other two being fairly user-friendly and intuitive – Unity's interface is very easy to learn and handle, and UE4 includes a new Blueprints visual programming system (see Fig. 1– Unreal Engine 4 blueprint example) that makes it easy for new developers to learn how to create interaction. The node-based Blueprints system allows the designers to use many tools that were previously only available to programmers, such as the creation of materials that update themselves according to the environment, random events, or procedural generation and scattering of objects, just by connecting nodes, events, functions and variables in a logical manner (Epic Games, 2017).

[2] CPU time represents the total amount of time a computer has been actively used for a given task (Free Software Foundation, Inc., 2017)

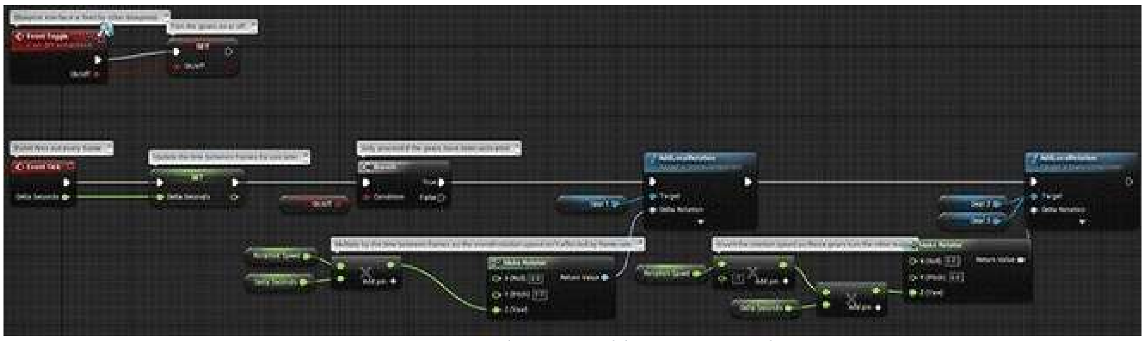


Fig. 1 - Unreal Engine 4 blueprint example

While Unity and UE4 are free to use commercially up to a predefined income where a royalty is added, CryENGINE uses a paid subscription model (Masters, 2014, paras. 10, 17, 21).

2.2.2. Pre-Rendering vs Real-Time Rendering

The 3D models are created in a computer, where the artist views them as a mathematical representation of points and surfaces in three-dimensional space. Rendering is the process that translates the modelled 3D assets into 2D images, taking into account the model's position, materials and the scene's lighting information to determine the colour of each pixel in the final image (Slick, What is Rendering?, 2017, 3-5).

Traditional rendering engines use many processes, techniques and algorithms that were designed for scenes with complex geometry, lighting and shapes, rather than for real-time performance. They also assume that thousands of samples will be taken in each pixel. Some of these design decisions make the engines less efficient at rendering simple scenes, leading to unnecessary work being done, depending on the scene.

These engines support a high range of geometric primitives. By using a ray tracer[3] that only supports simple shapes such as triangles, by tessellating[4] all the complex primitives before performing intersection tests, many calculations can be avoided, giving substantial advantages in performance, memory usage and reduced system complexity (Pharr, Jakob, & Humphreys, 2016, pp. 110, 136-139)

Rendering can be divided into two major categories: real-time rendering and offline or pre-rendering. Real-time rendering is mostly used in gaming and interactive applications, where images must be generated almost instantly depending on user input (Slick, What is Rendering?, 2017, paras. 7-10). Pre-rendered video can also be used for video games, when the processing power of the system is limited or the desired scene is too complex to be rendered in real-time (Kerlow, 2004, pp. 368-369).

"The ability to render polygonal models in real-time is almost as old as three-dimensional computer animation, but today's systems offer more realistic rendering and more complex models than what was possible a decade ago." (Kerlow, 2004, p. 120)

Real-time rendering allows the artists to preview the final output of the assets throughout the whole production pipeline almost instantly. This is a huge advantage, but it has its drawbacks. To be able to render in real time, the whole asset production pipeline must have

[3] Vd. chapter 2.3 – Physically Based Rendering for more information about the ray tracing technology

[4] Tesselation, in CG, refers to a process used to divide each polygon of a model into smaller parts. It is often used to divide each four-sided polygon (quad) in a model into two or more polygons with three sides each (triangle), which most graphics software and hardware are optimized to use (NVIDIA Corporation, 2017)

that end into account, in order to create optimized models by creating lower resolution geometry, textures, and less complex rigs, and reducing as much as possible the need for expensive calculations from the rendering engine (Beane, 2012, pp. 282-283). In order to create a fluid motion, a minimum of 18-20 frames must be rendered each second (Slick, What is Rendering?, 2017, para. 9). In real-time rendering, a lot of information such as scene lighting is often pre-computed (baked) to improve render speed. Static lights information is baked and mixed into static objects' textures, reflection maps can also be baked and simulations are pre-calculated and baked into geometry animation (Beane, 2012, pp. 284-285). There are many additional ways to decrease the required processing power in a scene: deleting or avoid modelling parts of models that will not be seen from the camera view, reusing assets as much as possible, creating instances of similar objects rather than copies (Cantor & Valencia, 2004, pp. 268-269), creating different levels of detail for the models and textures that will replace each other depending on the camera distance, or the use of billboards (image maps projected in a flat polygon surface) (Kerlow, 2004, p. 122).

Pre-rendering is used in situations where render speed does not affect the final product usability and/or photorealism is required, such as in the animation industry or visual effects. Each frame takes a long time to render, and higher resolution geometry is often used (Slick, What is Rendering?, 2017, paras. 11-13).

In 2013, the difference in graphics quality between pre-rendered cutscenes in videogames and the actual real-time rendered game was very noticeable. With the evolution of real-time rendering technology, videogames started using in-game graphic quality similar to the pre-rendered cutscenes they had in previous games. As stated by Gearnuke gaming news website writer Khurram Imtiaz (2014, para. 4) when comparing the game *The Last of Us for the Playstation 3* (2013) and the future release of *The Last of Us Part II* for the Playstation 4 (scheduled to be released in 2017), *"The large RAM of PS4 will allow them to finally kill the pre-rendered cutscenes, resulting in real-time gameplay and cutscenes."*

The convergence of various forms of media is the future of technology. Photorealistic 3D real-time rendering will eventually be used by industrial designers, architects, filmmakers, and other industries, which will lead to an increased sharing of information between fields (Sweeney *apud* Plante, 2015, paras. 6-8).

The use of game engines is also expanding to other industries, like film and architecture visualization. The hybridization of industries allows professionals to learn from each other, and often question themselves why they have not been using that technology all along. Despite that, a change on workflow in any industry requires time and effort, hence the reluctance of some companies to adapt to new technologies that were not originally designed for their industry, even if the software is more affordable than the one being used by the company (Rawn, 2015, paras. 2, 5).

2.3. Physically Based Rendering

"Fast becoming a standard in the games industry due to increased computing power and the universal need for art content standardization, physically based rendering aims to redefine how we create and render art." (Wilson, 2015, para. 3)

In the early years of computer graphics (CG), computers were expensive and had limited processing power and memory, which made physics simulation for rendering unfeasible.

With the evolution of technology, physically based approaches to rendering started to become viable. In the 1980s, physically based rendering techniques such as ray tracing, global diffuse lighting, and microfacet reflection models were introduced. At this time, CG started being used in the animation and film industry in movies such as *Star Trek II: The Wrath of Khan* (1982), *Tron* (1982) and *The Last Starfighter* (1984). The lack of computing power and memory made physically based processes like complex reflection models and global lighting effects unusable, and the use of "fake" lighting by strategically positioning invisible light sources in the scene to create more physically accurate results became an industry standard.

In the 1990s, early physically based rendering systems such as Ward's *Radiance* and Slusallek's *Vision* systems were released, leading to more interest and investigation in the area, producing more efficient algorithms and processes. In the late 1990s and early 2000s, the need for visual effects artists to incorporate realistic rendered images in video footage in the film industry led to the use of physically based approaches. In the animated film industry, photorealistic rendering is also used to create immersiveness, to make the viewer forget that he/she is looking at an environment that does not actually exist (see Fig. 2 – Life of Pi screen capture, before and after VFX). Some of the more modern rendering engines used in the industry today were released, such as Mental Ray, Arnold, and the new version of Pixar's Renderman with support for ray-tracing rendering.

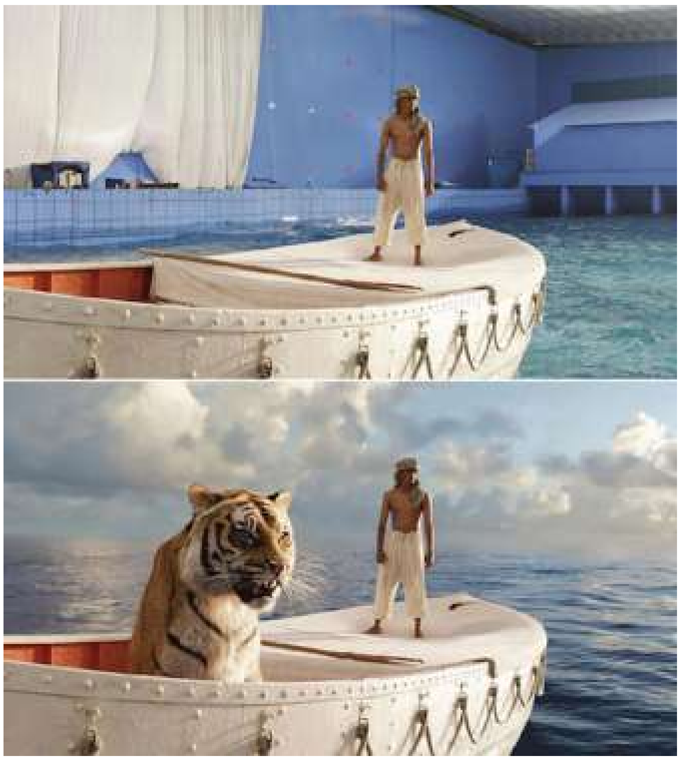


Fig. 2 - Life of Pi (2012) screen capture, before and after VFX

The term "Photorealistic Rendering" refers to the generation of images that are indistinguishable from photographs captured by a real camera, or that exactly reproduce the actual scene it is representing. Rendering processes, in the early years of the field, were focused on solving fundamental problems such as determining which objects were visible from a given viewpoint. As solutions to those problems were found and developed, and technology advanced, ideas from transversal fields such as physics, astrophysics, astronomy, biology, psychology, the study of perception and mathematics are increasingly included in rendering algorithms and methodologies (Pharr, Jakob, & Humphreys, 2016, pp. xix-xx).

The three most used rendering techniques are Scanline or Rasterization, Raytracing and Radiosity rendering. Scanline rendering performs calculations based on the assets' polygons instead of computing the results by calculating each pixel, and is often used in real-time rendering. Raytracing uses virtual rays of light that are emitted from the camera and bounce around the scene, interacting with the objects in its path and calculating the colour of each pixel in the final image (see Fig. 3 – Raytracing visual representation). Radiosity performs calculations without depending on the camera. The calculations are surface-oriented instead of calculating pixel-by-pixel. It is often used in conjunction with raytracing (Slick, What is Rendering?, 2017, paras. 17-18).

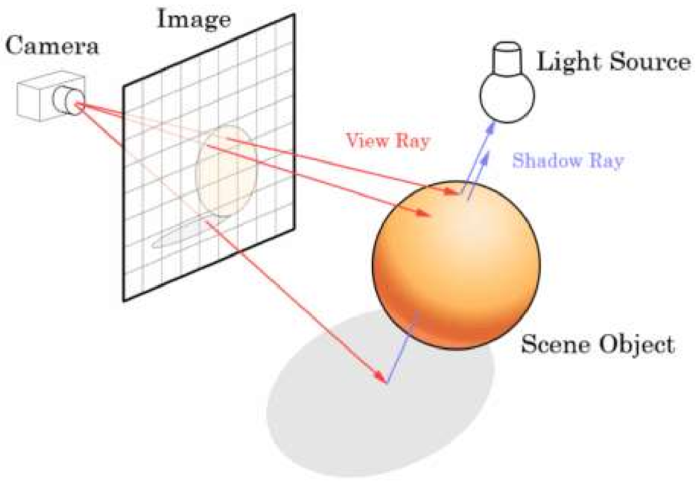


Fig. 3 - Raytracing visual representation

Most pre-rendering photorealistic rendering engines use the Raytracing algorithm, which involves following the path of a virtual ray of light though a scene, interacting and bouncing off of surfaces it comes in contact with. When it interacts with a surface, it takes into account the properties of the object at the intersection point, such as surface normal[5] or material properties, to calculate physical effects such as indirect light transport, and sometimes generating additional light rays originating at that point. (Pharr, Jakob, & Humphreys, 2016, p. 7-10)

Physically Based Rendering (PBR) is a rendering and shading method that uses a physically accurate simulation of how light interacts with surfaces in order to render assets as physically accurately as possible. In PBR, the assets are easier to create than with traditional shading methods, and they look accurate in all lighting conditions, even if they are created by different artists (Wilson, 2015, paras. 4-6). By using maps that are guided by physics principles, the need for the artist to guess and experiment with values is reduced. (McDermott, Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2, 2015, p. 3).

Artists with knowledge of the previous generation shading workflow already have most of the knowledge necessary to create content for a PBR system, needing only to learn the concepts and terminology of the new system. Some maps used in PBR have similar names to the ones used in the previous generation workflow, but they often are used in a different way (Wilson, 2015, paras. 8, 18).

[5] Vector perpendicular to a surface that defines the direction the surface is facing (Microsoft, 2017)

Very simple light sources like point lights do not exist in the real world. PBR is often based on *area lights*, a light source that is associated with a geometric object that emits light from its surface. Shadows are computed by casting a ray from the surface being tested, directly towards the light source – *shadow rays* – and checking if its path is being blocked by an object. In PBR engines, scenes with multiple lights are used as often as possible, since the contribution of each light can be computed individually and then added to each other to calculate the final result (Pharr, Jakob, & Humphreys, 2016, pp. 8-10).

PBR can be used for: *predictive rendering*, which is used in design and simulation with the objective of creating an exact reproduction of a scene without having to build it in real life; *plausible rendering*, which is used in entertainment and visualization applications to create results that look plausibly real; and *visually rich rendering*, where the result does not mean to be realistic, but uses the visual richness of reality (Shirley et al, 2013, p. 1). Using a PBR system does not mean that the resulting artwork will automatically be physically accurate, it just means that the calculations involved in the process obey the laws of physics. In a Frequently Asked Questions (FAQ) section on his PBR article, Marmoset lead artist Joe Wilson (2015, para. 17) commented that *"A great example of [stylized artwork] is Pixar's work, which is very stylized, yet often on the cutting edge of material accuracy."*

A typical rendering pipeline in real-time rendering is composed of three phases: user input, geometry processing and scan conversion. Each of this phases then include several substeps. In the user input phase, the user feeds information to the software. The positions of geometry are calculated. This phase's behaviour is controlled by the software developers. In the geometry processing phase, coordinate transformations, per-vertex shading and other geometry operations are performed to map the 3D information into a 2D representation. This is the phase that requires the most computing power. In the scan conversion phase, a rasterization process is performed in order to compute the color values for each pixel using the calculations performed in the previous phase (Bao & Hua, 2011, pp. 9-10).

2.3.1. Light Mechanics in PBR

To create realistic materials, it is important for 3D artists to understand that light behaves in different ways when traveling through different media or interacting with different surfaces.

When light hits a surface, it can be reflected off of it (reflection), and/or penetrate the surface (refraction). When light is reflected, it does not transpose the surface – instead it bounces off in an opposite angle relative to the surface normal. When light transposes the surface boundary, entering the illuminated object or medium, it is scattered inside of it, where it can be absorbed completely, or exit the object before that happens. This is known as "diffuse light", "diffusion" or "subsurface scattering" (Russell, 2015, paras. 4-6). When light is absorbed, its intensity decreases, but the direction of the ray stays the same, and when it is scattered, the intensity stays the same, but the ray direction is changed. As stated by Allegorithmic technical artist Wes McDermott (2015, p. 3), *"The further light travels in such a medium/material, the more it is absorbed and/or scattered. Therefore, object thickness plays a large role in how much the light is absorbed or scattered."*

The light interacts not only with objects, but also with *participating media* in the environment, such as smoke, fog or dust. These media can absorb or scatter light in different directions – as happens in the real world. When light interacts with volumes (Volume Scattering), the engine uses a probabilistic process instead of creating calculations for each individual microscopic particle. With this process, effects such as atmospheric haze, beams of light and subsurface scattering can be accomplished (Pharr, Jakob, & Humphreys, 2016, pp. 13, 575).

In materials with wider scattering distances, like skin or wax, we can often see light scattering out of the back side of the object. These objects can be called translucent. In some materials, almost no scattering is evident, and light can travel almost unaffected through the medium, as happens in clear glass (Russell, 2015, para. 8).

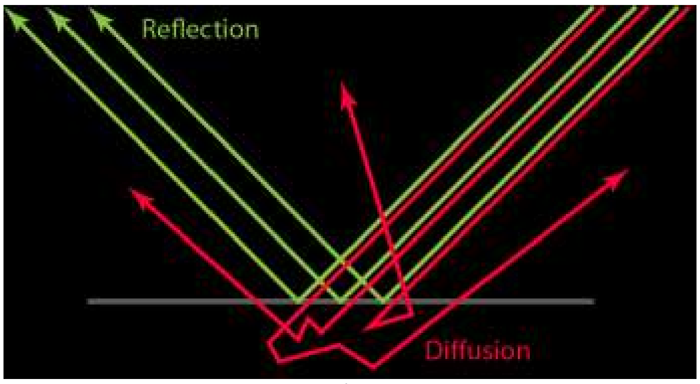


Fig. 4 - Light interaction

Specular reflection is light that is reflected when it hits a surface. Diffuse reflection is light that is refracted, scattered inside the medium, and refracted out again (see Fig. 4 – Light interaction). Diffuse materials are usually very absorbent. When light travels for too long inside them, it either is completely absorbed or it exits the object at approximately the same point it entered it, so the distance between the entry and exit point is usually neglected in the calculations – unless rendering materials with high scattering but low absorption using Subsurface Scattering, where that distance is taken into account (McDermott, Light and Matter: The Theory of Physically-Based Rendering and Shading, vol. 1, 2015, p. 4).

The orientation of the surface is defined by 3 elements: the shape of the mesh, on a larger scale; the normal map[6] being applied to it, for large details; and the roughness or glossiness map[7], to represent microfacets[8]. Calculating the real surface direction at a microscopic level for each light ray that hits it would need too much memory and computing power, so instead, the roughness map is used to describe a general measure of roughness instead of the actual microsurface details. Microsurface detail has a large effect on light behavior (see Fig. 5 – Light interacting with microsurface detail), and is a very important characteristic of PBR, since in the real world, there is a wide variety of microsurface features. An artist can paint variations of the microsurface roughness directly on the roughness/glossiness map, and PBR will display the change in reflection shape and its relative intensity. Since the reflectivity and microsurface detail values are physically related, it is an advantage for physically accurate rendering that the artist does not have independent control of each value. Marmoset founder and engineer Jeff Russell (2015, para. 36) also states on his article about PBR basic theory that *"Microsurface properties have other subtle effects on reflection as well. For example, the 'edges-are-brighter' Fresnel effect diminishes somewhat with rougher surfaces (the chaotic*

[6] Color image used to control the direction of each vertex's normal vector in a 3D model (Polycount, 2016)

[7] Vd. chapter 3.6.1 – PBR Workflow for more information on each map type and usage

[8] microscopic details on a model's surface

nature of a rough surface 'scatters' the Fresnel effect, preventing the viewer from being able to clearly resolve it)."

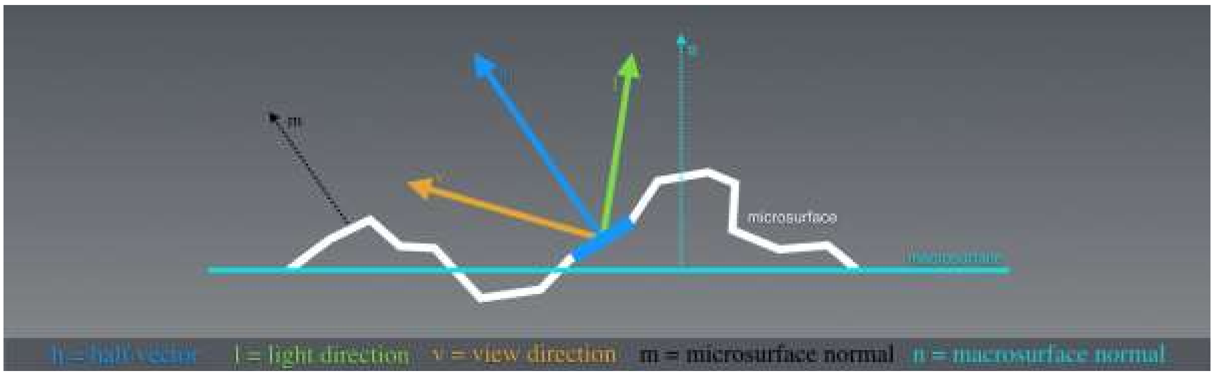


Fig. 5 - Light interacting with microsurface detail

The color of an object is defined by the light wavelengths that are reflected by the surface. The remaining wavelengths are absorbed by the object. The specular highlights on dielectric (not electrically conductive) materials is almost independent of wavelength, so it is usually the same color of the light source (McDermott, Light and Matter: The Theory of Physically-Based Rendering and Shading, vol. 1, 2015, p. 6).

Reflection and diffusion are mutually exclusive – if a light ray is reflected, it does not penetrate the surface, so it cannot be diffused. This means that a material with high reflectivity will show little to no diffuse light, while materials with high diffusion cannot be very reflective. This happens because of the Energy Conservation principle, which states that the intensity of the light leaving a surface must never be higher than that of the light received by that surface. When using PBR, the Energy Conservation principle is always enforced by the shader, which is an advantage for the artists working with it – *"(...) [Energy Conservation] allows the artist to work with reflectivity and albedo values for a material without accidentally violating the laws of physics"* (Russell, 2015, para. 11).

A BRDF (Bidirectional Reflectance Distribution Function) is a function that defines the reflectance properties of a surface. For it to be physically plausible, it needs to use the Energy Conservation principle. (McDermott, Light and Matter: The Theory of Physically-Based Rendering and Shading, vol. 1, 2015, pp. 6-7).

The Fresnel effect states that light that lands on a surface at a grazing angle will show more reflectivity, which will make an object appear to have brighter reflections near the edges. For all smooth materials, reflectivity becomes gradually closer to 100% as the surface angle approaches 90º from the subject's point of view, or as defined by Augustin-Jean Fresnel, the french physicist after whom the effect was named, *"(...) the amount of light you see reflected from a surface depends on the viewing angle at which you perceive it."* (McDermott, 2015, p. 7).

PBR reduces the artist's control over the Fresnel effect, to create physically accurate representations. It uses material properties such as gloss and base reflectivity to automatically calculate the reflectivity of the material at a given angle (Russell, 2015, paras. 21-23).

PBR materials can be roughly divided into two categories: metals and non-metals (with exceptions). Dielectric materials need an albedo (diffuse) and specular map. The color from metals comes from the reflected light so they do not need a diffuse color, but they need an RGB specular map. Painted metal or rusted parts of metals are treated as a dielectric material. It is important to add that in PBR, every color-related calculation is made in Linear Space, with the gamma value set to 1.0, so that light mimics its real-world behavior. To be displayed on a screen, the image must be in Gamma-encoded Space (sRGB) (McDermott, Light and Matter: The Theory of Physically-Based Rendering and Shading, vol. 1, 2015, p. 11). Electrically conductive materials will often exhibit reflectivities as high as 60-90%, where insulator (dielectric) materials usually have very low reflectivity, in the 0-20% range. Russell (2015, para. 16) adds that *"It is this duality between metals and just about everything else that leads some rendering systems to adopt 'metalness' as a direct input. In such systems artists specify the degree to which a material behaves as a metal, rather than specifying only the albedo & reflectivity explicitly".*

2.4. 3D Animated Film Production Technologies

With the evolution and increased accessibility of 3D animated film production technologies and tools, more people can experiment and learn how to create and use 3D assets for films and game development. With increasingly more capable and user friendly software and hardware, more quality indie games and short films are being produced, allowing for smaller teams such as in the short film Adam (2016)[9] to create projects with a quality level that rivals big animated film production companies.

"These days, children as young as five are playing with [3D Art] and older people who could barely draw stick figures can bring to life the beauty that has been in their imagination. Solo artists have already been able to create short animations on their own. Perhaps in the near future, individuals may be able to make entire movies all by themselves." (Chopine, 2011, p. 12)

2.4.1. Modelling and Sculpting

Modelling and sculpting are the most common methods used to create 3D assets for animation films. Modelling allows for more control over the individual elements of the 3D mesh, where the artist is able to control each vertex individually when needed. Sculpting gives the artist more freedom to express himself, by mimicking the tools used in physical sculpting to a virtual environment, where the artist can often disregard technical modelling details like geometry, topology and resolution.

Polygon modelling is the most widely used approach in animation and video game industries. A polygon is a virtual element defined by a minimum of three vertices connected by edges (a triangle). Polygon meshes render quickly and are supported by most 3D software packages, which means they can easily be imported and exported between them (Cantor & Valencia, 2004, p. 259).

"One potential disadvantage of polygon models is that they have a tendency to look faceted, so to create curved and smooth surfaces you need to increase the surface's resolution by using a large number of vertices" (Cantor & Valencia, 2004, p. 259).

Box modelling is the most common polygonal modelling technique, where the artist starts with a low-resolution primitive geometry object (often a cube), that they will modify by moving its components until they get the desired result. Depending on the situation, the final

[9] Vd. chapter 2.5.2 – Adam (2016) – Unity Technologies for more information on the production of this short film

result may or may not be subdivided to smooth the hard edges (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 6-8).

Edge modelling is also a polygonal modelling technique, often used to create complex models that require a specific topology and edge flow, as it happens when modelling a character's face (see Fig. 6 – Facial edge flow in 3D modelling). This is accomplished by using polygons to create outlines of each prominent part of the object, and then filling the gaps between them (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 9-10).

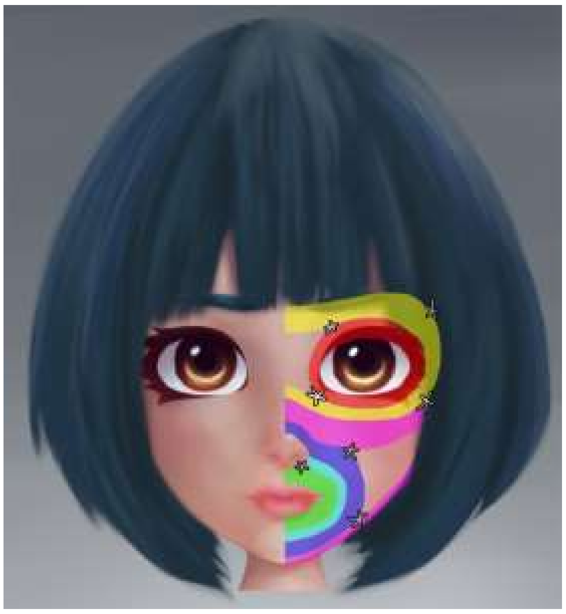


Fig. 6 - Facial edge flow in 3D modelling

NURBS modelling is used for automotive and industrial modelling. NURBS models are not polygon based. They are created by automatically generating geometry connecting two or more Bezier curves (splines), which creates smooth surfaces, and instead of faces, edges and vertices, the geometry is controlled by moving the splines' handles (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 11-12).

Digital sculpting is a technique that gives the sculptor artistic freedom by allowing them to ignore modelling constraints such as topology and edge flow and use real-world sculpting techniques in a digital environment. It is often done with a graphics pen tablet where the artist can interact with virtual chunks of clay and create high resolution meshes (often with millions of polygons) which allow for the creation of very small surface details (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 13-14).

Procedural modelling uses mathematical algorithms to generate models. It is often used to generate organic models such as trees or foliage, or complex environments such as cities or landscapes, where there can be a great variation that would be too time consuming to create by hand. Some examples of software that use this technique are Vue, Bryce, Terragen, SpeedTree and CityEngine (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 15-17).

Image Based modelling generates 3D models from a set of two-dimensional images. This technique has been used in movies such as *The Matrix* where time or budget restrictions do not allow for the manual creation of a full 3D model (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 15-16).

3D Scanning, as the name implies, uses 3D scanner technology to generate an accurate mesh from a real-world object. For this technique to be used, a real object must exist before the model is created, which applies only to specific situations (Slick, 7 Common Modeling Techniques for Film and Games, 2016, paras. 17-18).

Most of the modelling packages used in the industry are equally capable, with some being arguably better than others in specific tasks. Each artist should experiment with as many applications as they can in order to find which is better for their needs or which they like working with the most.

One of the most important software aspects for aspiring animation filmmakers is the cost of the software. Short film filmmakers often have a low budget to buy software, rent a studio, equipment and hire a crew, and everything else that is needed in a short film production. Buying expensive software can often be impossible, so many studios opt in using free software or inexpensive indie versions when they are available (Manasseh & Ephraim Studios, 2014, para. 1).

Autodesk Maya and 3ds Max have been the industry standard modelling and animation software for many years. Maya is arguably the best of the two for animation, while 3ds Max is often considered better for modelling and asset creation for game development (Fronczak, 2011, para. 5-6). 3ds Max has been used in video games such as the Grand Theft Auto series, and Maya was used in multiple films such as the Spider-Man and Lord of the Rings series, and is often used in animation films (Smit, 2012, paras. 5-9).

Blender is a free to use commercially, all-in-one package, capable of most of the features offered by the main paid software, and has a large online community with many free courses and tutorials available (Fronczak, 2011, para. 4). It has been used in movies, commercials and games, including some open projects created by the Blender Foundation such as *Big Buck Bunny* (2008) and the game *Yo Frankie* (2008) (Smit, 2012, paras. 12-14). *"Their site [Blender] is also full of resources for beginners and experts and the community is warm and friendly with a lot of great insight and tips."* (Smit, 2012, para. 12).

2.4.2. Rigging

"Rigging is the process of setting up a controllable skeleton for the character that is intended for animation." (Chang, n.d., para. 50)

For a 3D model to be animated, unless it is a simple animation that does not require deformation on the mesh, it needs to be rigged.

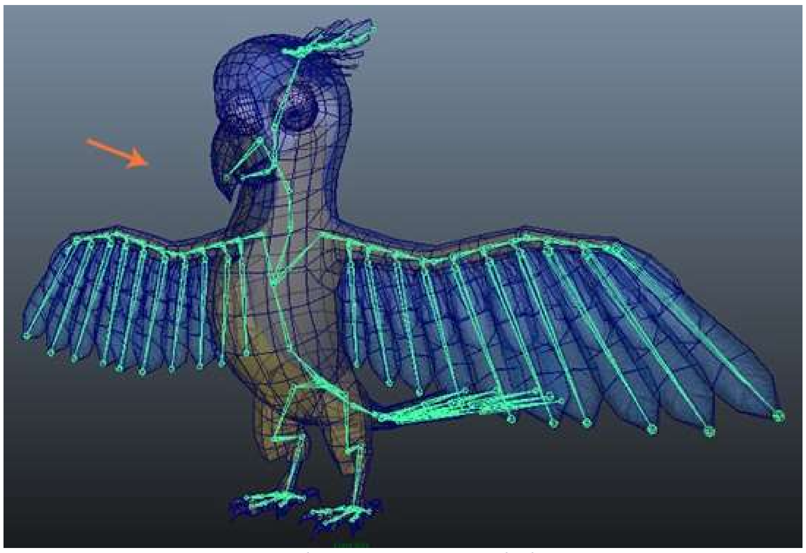


Fig. 7 - Character rig in Autodesk Maya

Rigging is the process of creating an (usually) internal virtual skeleton composed of joint deformers (see Fig. 7 – Character rig in Autodesk Maya) that will then be animated and will cause the model geometry to deform. These joint deformers work similarly in most 3D applications (Chopine, 2011, p. 87). Without a rig, the 3D model can be animated by translating, rotating or scaling it, but the mesh will always be stuck in the same pose it was created in (Pluralsight, 2014, para. 1).

"Joints are the points of articulation you create to control the model." (Pluralsight, 2014, para. 2).

A model's rig is a system of joints, bones and control handles following a parent-child hierarchy, which allows the animators to pose and animate the model (Slick, What is Rigging?, 2016, paras. 2-9). Joints in a humanoid 3D model are positioned roughly on the same places our skeleton's joints are. Generally, the joints will be positioned on a point where the model will rotate or bend, such as an elbow or shoulder (Pluralsight, 2014, para. 2).

After the rig is completed, it is bound to the 3D mesh by a process named skinning (Slick, What is Rigging?, 2016, paras. 2-9). Without skinning, the mesh will not be affected by the joints movements. After skinning, the relationship between joints and mesh is often not ideal, so the rigger uses weight painting, which is a technique that sets the influence (weight) each joint will have on each of the mesh's vertexes' movement (Pluralsight, 2014, paras. 9-10).

There are two main techniques used by a rig that define the way the joints' hierarchy will affect the rig's movement: Forward Kinematics (FK) and Inverse Kinematics (IK).

FK gives more control over the joints hierarchy, but since each joint needs to be positioned independently, it is not ideal in some situations where IK would be more efficient (Pluralsight, 2014, para. 6). In FK, rotating a joint in the rig will also affect all the joints that are below it in the joint's hierarchy. As an example, in a humanoid rig, when the shoulder joint is rotated, the whole arm – elbow, wrist, hand, etc. – will rotate with it. When using this technique, the animator should follow the hierarchy and move the joints that are higher in the hierarchy first, in this case, start with the shoulder, then elbow, then wrist, and so on (Slick, What is Rigging?, 2016, paras. 10-11).

When using IK, the process is reversed: the last joint in the hierarchy of the IK section is the one that dictates the position of the joints above it. This means, using the same example as above, that the animator will position the wrist of the character in the desired place, and the position and rotation of the elements above it (elbow and shoulder) will automatically be calculated by the rig. This technique is often used when the final joint of the hierarchy needs to be placed precisely at one specific location (Slick, What is Rigging?, 2016, paras. 12-14).

"[Inverse Kinematics] allows the animator to animate independently of the chain's hierarchy. Because of this IK is great when needing to have a character's arm stay planted on something." (Pluralsight, 2014, para. 5)

When rigging, most of the rig's joints are positioned inside the mesh. Controlling the joints directly would be inefficient and cumbersome for an animator, so the rigger creates control curves, which are visual elements, usually positioned outside the mesh around the controllable joins (see Fig. 8 – Rig control curves), so that the animator can easily select and manipulate them (Pluralsight, 2014, para. 7).

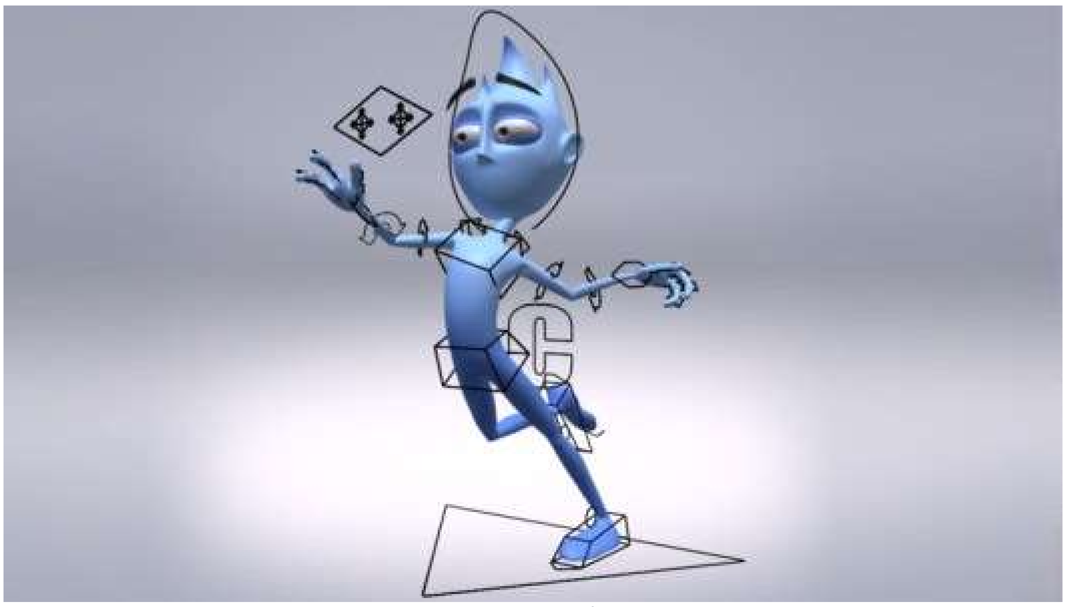


Fig. 8 - Rig control curves

Joint Constraints are limits that can be imposed to an object's position, rotation or scale. As an example, using a parenting constraint on two objects, makes the child object always follow the transformations applied to its parent (Pluralsight, 2014, para. 8). Constraints can be used to restrict the rig's movements, in order to prevent unrealistic animations such as the elbow bending backwards or the head tilting or rotating too much, or to prevent or control the use of techniques such as squash and stretch, which can deform the mesh in unrealistic ways when exaggerated – although that may be desired, depending on the artistic style of the animation (Slick, What is Rigging?, 2016, paras. 15-17).

The facial rig for a character is usually separated from the rest of the body's rig. Facial rigs often use a mixture of joints/bones and blend shapes that directly deform the mesh (Slick, What is Rigging?, 2016, para. 18). When using blend shapes, the artist modifies the mesh directly and saves multiple poses that he will later be able to control using sliders (Pluralsight, 2014, para. 4).

2.4.3. Animation

In 3D, animation consists of varying the properties of an object through time, such as position, rotation, shape and surface style, in a three-dimensional space (Rebelo, 2003, para. 1).

Modern 3D software has the ability of keyframe interpolation (see Fig. 9 – Animation interpolation), which takes into account the transformations that happen to an object between one keyframe and the next one, and automatically creates the movement in between the two frames (Gray, 2015, para. 6).

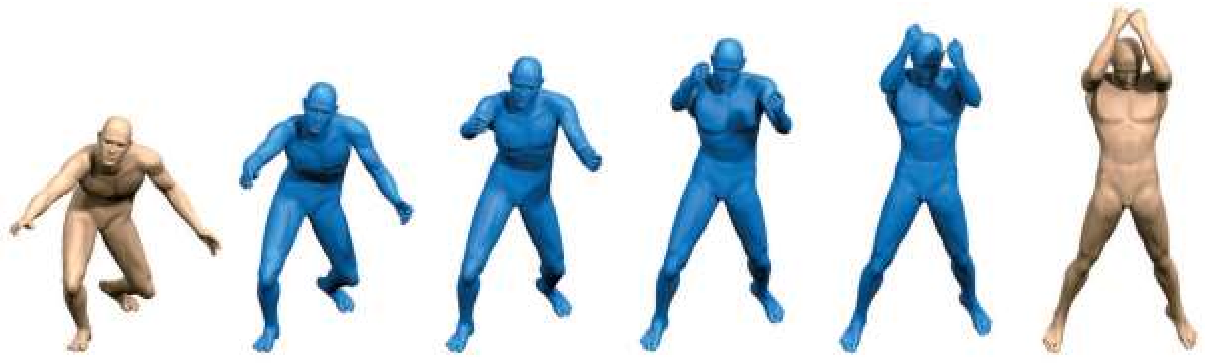


Fig. 9 - Animation interpolation

Motion Capture (MoCap) can produce realistic results in animation. It uses cameras and specific suits with markers that have their position tracked by infrared cameras and translated into movement data that can be interpreted by the 3D software and mapped on to the character's rig (Rebelo, 2003, para. 9). At first, MoCap was seldom used for animation due to the technology's limitations, but is now used in many 3D animation scenarios, from video games to CG effects in movies (Gray, 2015, para. 7).

MoCap is a costly technology, to which not everyone has access to. It is also a technology that requires a lot of cleaning up after the capture process is finished to correct capture errors or make it more believable, particularly when the target character is not a humanoid character. Artists that don't have access to a MoCap studio often learn how to incorporate this technique by importing MoCap data from online MoCap libraries (Gray, 2015, paras. 8-9).

"MoCap *subjects, usually actors, are placed in a special suit containing sensors that record the motion of their limbs as they move. The data is then linked to the rig of a 3D character and translated into animation by the 3D software."* (Gray, 2015, para. 7).

One of the most important skills of an animator is the capacity to observe the world around them. Learning from film, theatre, comic books and other animators' work is also important, and some animators also take acting lessons (Gray, 2015, paras. 10-11).

3D animators, while often not in charge of the character's rigging, should have a good understanding of how rigs and FK and IK systems work. Animating is a craft that requires a lot of practice and attention to detail (Gray, 2015, paras. 13-14).

To create realistic animations or real-world behaviours in animation, tools like physics engines can also be used to apply physical laws (such as gravity) to an object to generate its movement (Rebelo, 2003, paras. 6-7).

2.4.4. Textures and Maps

A few years ago, the need for optimization was much higher than today. Textures had to be very small and with limited color palettes, geometry resolution had to be extremely low and even when using many optimization techniques, the results would not even approximate those we can achieve today.

Textures can be created by using image editing software such as Adobe Photoshop or 3D painting software such as Mudbox or ZBrush (Beane, 2012, p. 39).

Maps such as displacement, normal or bump maps can be used as an addition to the assets' modelling by creating or simulating surface details that would be too time consuming to create using standard modelling tools (Cantor & Valencia, 2004, p. 264). Bump maps are grayscale maps that simulate height on a model's surface, creating the appearance of depth in the direction of the camera. Normal maps are RGB color maps that simulate height on all three axes, affecting the angle of the geometry normals directly, giving more accurate results. Displacement maps are grayscale maps that affect the geometry directly, which gives the best results, but requires a high amount of processing (Chopine, 2011, pp. 159-160).

When creating maps for real time, their limitations are defined by the system specifications. For optimization, multiple techniques can be used, such as tiling textures so that the final result appears to be a single large map (Kerlow, 2004, p. 234).

Software such as The Foundry's MARI, Maxon Bodypaint 3D, Pixologic ZBrush, Autodesk Mudbox, Pilgway 3D-Coat and Allegorithmic Substance Painter allow for the painting of textures directly in the 3D model (sandy, 2014).

Adobe Photoshop or Gimp are often used to paint or edit textures in a 2D space, directly over the exported UV Maps (sandy, 2014, p. 1, paras. 42-46). Photoshop's versions CS6 and CC can now also import 3D asset files into the software and create new textures for it, which can be previewed directly on the 3D model as an UV overlay. The textures can then be edited or painted directly in the 3D model, or as a 2D image. They also allow for texture map reparameterization, which corrects distortion caused by poor UV mapping, creation of tiles for repeating textures, and for the selection of specific parts of the model to be painted or temporarily hidden for ease of use (Adobe Systems Incorporated, 2017).

Knald and xNormal are applications used to generate normal, ambient occlusion and displacement maps from high resolution meshes. Other applications such as Bitmap2Material allow for the generation of those maps from image files (sandy, 2014, p. 2, paras. 1-14).

UVLayout and Ultimate Unwrap 3D are UV Mapping applications, used when the standard UV Mapping functionalities of the main software aren't enough (sandy, 2014, p. 2, paras. 15-23).

2.4.5. Lighting and Rendering

Aldric Chang, founding Managing Director of Mediafreaks defines lights in 3D as *"(...) objects that are designed to simulate how lighting works in real life."* (Chang, n.d., para. 56).

High Dynamic Range Images (HDRI) are spherical images that are often used as a sky dome where light is emitted from the image to illuminate the scene (see Fig. 10 – HDRI example). They allow for realistic renders and for incorporation with live action footage, to match the original video lighting (Chopine, 2011, p. 188).

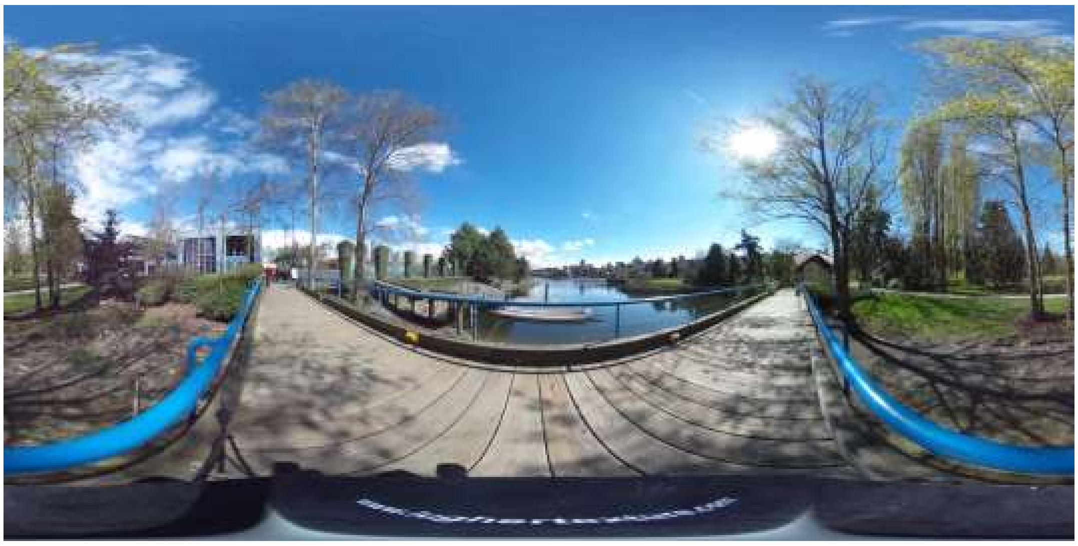


Fig. 10 - HDRI example

"An HDRI contains not only color information, but also information about the luminosity or intensity of the light." (Chopine, 2011, p. 188)

Light animation effects can be used to create or change the mood in a scene. Light produced by natural phenomena such as lightning or fire usually require animation to produce realistic results. These animations can either be created by animating the light's parameters, using procedural animation, lighting "tricks" as used in traditional stage plays or movie special effects, or added in post-production (Kerlow, 2004, pp. 318-320).

"A wide variety of lighting effects that affect the mood of a scene can be created by animating the intensity of a light source as well as its color, cone angle, and fall-off." (Kerlow, 2004, p. 317)

The rendering setup process varies from engine to engine. Most traditional rendering engines are able to render in passes (see Fig. 11 – Render passes), which is a process that breaks a render into individual pieces, such as color information, shadow information, highlight information or reflection information. By using this method, the compositing artist has more control over the final look of the scene, since each of these pieces can be individually edited and then merged together in the end (Beane, 2012, pp. 244-245).

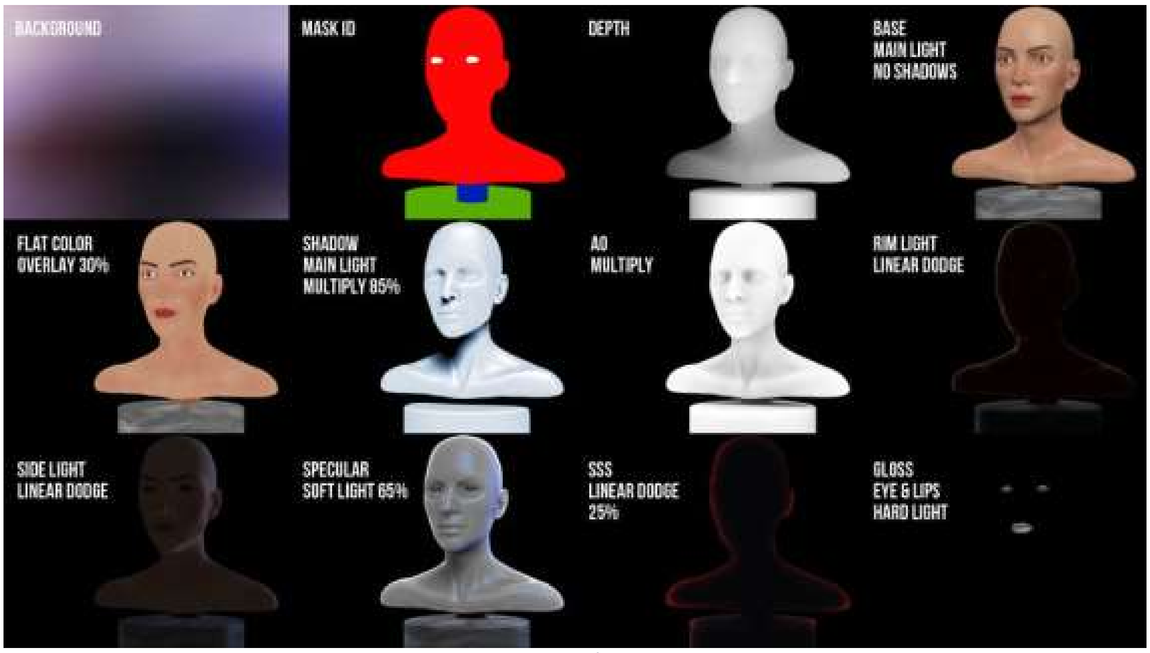


Fig. 11- Render passes

Non-photorealistic rendering techniques can be based on three-dimensional rendering techniques or two-dimensional image processing techniques (post-processing filters). These techniques are often called *toon shaders.* With these techniques renders like traditional media simulation such as watercolor or dry brush effects can be accomplished (Kerlow, 2004, pp. 170-171).

"Non-photorealistic rendering techniques became increasingly popular during the late 1990s to create three-dimensional computer animation and still images that look as if they were created with traditional techniques." (Kerlow, 2004, p. 170)

"The OpenEXR file format was initially developed as a proprietary format at Industrial Light & Magic but released as an open source format in 2003." (Kerlow, 2004, p. 421)

OpenEXR is an image file format often used in traditional rendering. It is a High Dynamic Range format that can store information using up to 32-bit floating-point data, and uses the concept of "Multi-Part" files, which means that one EXR file may contain multiple separated images, which allows for multi-pass renders to be stored in one file only (Industrial Light & Magic, 2017).

Post-production and compositing software such as The Foundry's NUKE support the OpenEXR file format, which is often preferred for its multi-channel support.

2.4.6. Post-Production and VFX

Autodesk software is widely used in VFX and animation studios worldwide. Software packages such as Maya, 3D Studio Max and Motionbuilder have been used by most big studios' pipeline in movies that were nominated for the Oscar of Best Visual Effects at the Academy Awards. Some examples of such movies were *Harry Potter and the Deathly Hallows Part II* (2011), *Hugo* (2011), *Real Steel* (2011) and *Rise of the Planet of the Apes* (2011) (Wolfe, 2012, paras. 3-6).

Animation studios such as Dreamworks Animation and Pixar also use Autodesk Software in their Oscar nominated animation films – short and feature – such as *Kung Fu Panda* (2008), *Puss in Boots* (2011) and *La Luna* (2011) (Wolfe, 2012, paras. 8, 11).

Some Visual Effects can be created in 2D software, but effects such as hair, fur and object destruction are often simulated using 3D physics simulation engines that generate physically accurate animations based on factors such as gravity or forces interacting with the objects. These engines can simulate particle systems, hair and fur, fluids, and rigid or soft bodies (such as falling debris or clothes, respectively). Particle generators generate a number (often thousands) of points in three-dimensional space that react to fields or forces defined by the artist. After these points are generated and animated by the engine, any type of geometry or shader can be assigned to them. Using particle systems allows the artist to create animations of multiple objects at the same time, which would require a lot of time and effort to animate individually, as would be the case when trying to animate a swarm of bees, rain drops or a crowd of people (Beane, 2012, pp. 214-215).

Fluid simulation is a type of particle simulation specifically designed to simulate the behaviour of fluids. These engines can also be used to simulate other media that behaves in a fluid-like manner, such as smoke or fire. Often simulations are generated in an external software such as RealFlow and then exported as a geometry or particle cache (Beane, 2012, pp. 218-219).

Hair and fur simulation and rendering is difficult to master. With these engines, the hairs can be styled and properties assigned to them, such as the color, look and behaviour, which will allow the engine to generate its animation (Beane, 2012, pp. 216-218).

Rigid and soft bodies' simulation are used to simulate the behaviour of objects on collision. Rigid bodies are objects that do not deform, such as falling debris, object shattering or ragdoll animations, while soft bodies are objects that can deform, such as clothes, muscles and fat (Beane, 2012, pp. 220-226).

The Foundry's NUKE is a VFX software with a node-based workflow used for most post-production tasks such as compositing, rotoscoping, color-grading, etc. (Creative Bloq Staff, 2014, paras. 1-5). It supports Python programming, which is the language most common in 3D software (Creative Bloq Staff, 2014, para. 17).

SideFX Houdini is an all-in-one package that includes powerful animation and VFX tools, such as particles and smoke simulation. It is used by companies such as Dreamworks Animation, Blue Sky Studios, Image Engine and Axis Animation (Smit, 2012, paras. 2-4).

Next Limit Technologies RealFlow is an environment and physics simulator, mostly used for fluid simulations (Fronczak, 2011, para. 8).

Lightworks is a professional non-linear editing software, used in some major films such as *Pulp Fiction* (1994), *Gangs of New York* (2002), *28 Days Later* (2002), *The King's Speech* (2010), *Hugo* (2011), among others. It has a paid version and a free version with some restrictions in output formats, resolution and plugins available (Manasseh & Ephraim Studios, 2014, paras. 9-10).

2.4.7. Other Technologies and Techniques

Often external 3D software or plugins are used for terrain generation, vegetation and other elements. Some techniques such as heightfield maps generated procedurally or from geographical data are used to create terrains (Chopine, 2011, pp. 207-211).

"Most of the 3D modeling packages out there are ill equipped to create terrains with much complexity. However, there are several standalone applications and plugin tools that can generate terrains, and other natural and urban scenary elements, very well." (Chopine, 2011, p. 207)

"Alembic is an open computer graphics interchange framework. Alembic distills complex, animated scenes into a non-procedural, application-independent set of baked geometric results." (Lucasfilm Ltd. & Sony Pictures Imageworks Inc., 2017, para. 1)

Alembic is a format used to efficiently store 3D geometry. It stores only the vertex positions and its variation over time, and discards any rigs or any procedural information used to create the geometry. It is often used for complex meshes or to store finalized procedural or simulated animations, in situations where using the resulting geometry, with all its resolution and the amount of information attached to it, would make it too "heavy" to be handled by the target 3D application (Lucasfilm Ltd. & Sony Pictures Imageworks Inc., 2017, paras. 1-6).

Scripting can also be a very useful skill for a 3D artist. It can help in creating random events such as light flickering, or automating repetitive tasks. The most common scripting language in 3D applications today is Python (Chopine, 2011, p. 243).

2.5. Real-Time Rendered Short Films

Usually, animation films over 30 minutes are considered long films. Many short animations under one minute are created for TV commercials, experimental pieces or student projects (Kerlow, 2004, p. 44).

"There is no standard length of time that determines what is a short or long piece of computer animation. Usually anything over 30 minutes is considered long." (Kerlow, 2004, p. 44)

In the late 1990s, the animated short film industry produced some of the best short animation films, including *Geri's Game* (1997), *Bunny* (1998), *Bingo* (1998), and *Tightrope* (1999), using traditional rendering methods (Kerlow, 2004, p. 28). With the new real-time rendering technologies' development in the last few years, real-time rendered short films started being developed and released.

2.5.1. The Gift (2016) – Marza Animation Planet Inc.

MARZA Animation Planet Inc. Studio, known for their work in videogame industry for Sega Games and in 3D animation films such as *Space Pirate Captain Harlock* (2013) created the short film *The Gift* (2016) as a proof of concept for a Unity plugin they developed in order to improve the CG animation workflow (Sanchez, 2016, paras. 1-4). The plugin allowed for alembic import[10], which created the possibility of rendering huge amounts of geometry at the

[10] Vd. chapter 2.4.6 – Other Technologies and Techniques

same time (see Fig. 12 – Marza's tests on the Alembic Importer for Unity), and OpenEXR[11] image capture. The source code for both features is now available on GitHub as Open Source (Milligan, 2016, para. 6).

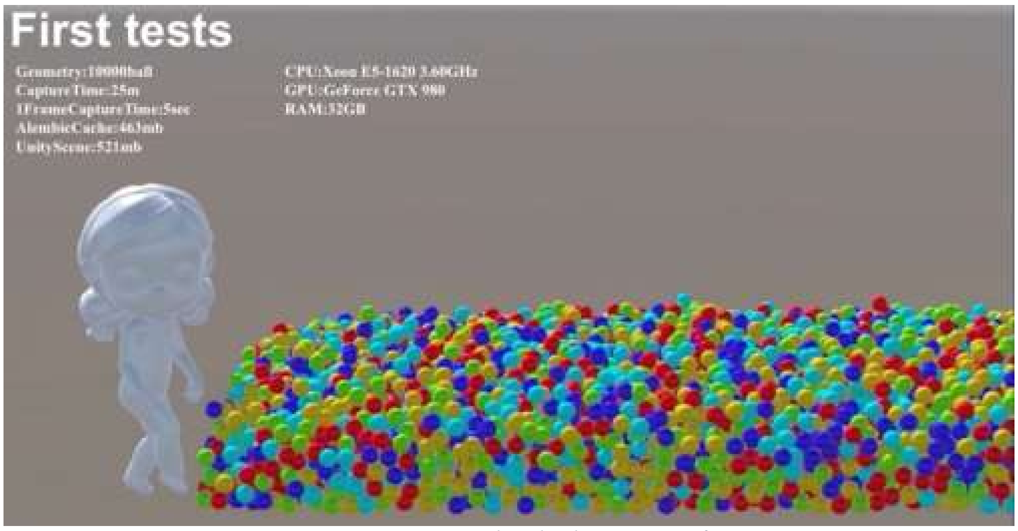


Fig. 12 - Marza's tests on the Alembic Importer for Unity

Traditional rendering methods must deal with lengthy timelines, expensive hardware and limited review processes. The team at MARZA included the Unity game engine in their workflow to avoid these issues, while maintaining feature-quality character motion and shaders. With this workflow, the team also reduced their production time in 20-30% (Milligan, 2016, para. 5), which allowed the artists to focus more on the quality of the work instead of the production deadlines.

By using this workflow, even in the early stages of production, the images being rendered had a visual quality very close to the final product, with the renders being much quicker than those made using traditional rendering methods (Sanchez, 2016, para. 6).

Hiroki Omae, MARZA's Regional Director stated that one of the most critical factors that made this possible was Unity's addition of Physically Based Rendering technology to the engine (Milligan, 2016, para. 7).

2.5.2. Adam (2016) – Unity Technologies

"Game engines are changing the way cinematics can be created and opening the door to new forms of CG entertainment" (3D World, 2016, p. 19)

The short film *Adam* (2016) was created using only the Unity built-in functionality as a proof of concept to show the potential of Unity in filmmaking and its versatility (3D World, 2016, p. 20), by using the newly developed technologies to be included in the next releases of Unity, such as Real-Time Area Light technology, which was presented by Unity at Siggraph 2016 (Muller, 2017, para. 12). External software such as 3D Studio Max, Character Studio, MotionBuilder, Substance Painter and Quixel was used for the creation of assets and animation (3D World, 2016, p. 21).

[11] Vd. chapter 2.4.5 – Lighting and Rendering

Fig. 13 - Adam's screenshot inside Unity

Most of the plugins or packages used in the short film *Adam,* such as the Volumetric Lighting package or the Alembic plugin are now available to the public or will be included in Unity's future releases (Muller, 2017, paras. 8, 10).

By combining filmmaking and game-making processes, all areas of production could work in parallel with each other, and because of the nature of real-time rendering engines, more experimentation with lighting and camera placements was possible. The Visual Sequencer, also known as Storyteller Tool was one of the main tools used in the production of this film (3D World, 2016, p. 23). While it is not yet available to the public, it is scheduled to be made available in one of the next releases of Unity.

In the production of *Adam* (2016), Plamen "Paco" Tamnev, the character and environment artist, started by creating the high-resolution geometry models in ZBrush, which then were re-created as a low resolution version in 3ds Max. After creating the UVs[12], additional maps' details and materials were created and baked in Substance Painter (Bagard, 2016, para. 5).

Substance Painter 2 was the chosen software for the texture creation because of its intuitive interface, ability to quickly try different ideas for a given material, and mainly because of its good integration with Unity, which allowed the artist to see, inside Substance, the same results he would have after exporting the textures into Unity (Bagard, 2016, para. 6). This tool was also used to add additional materials to the normal maps generated from the higher resolution model (Tamnev, 2016, para. 10).

Marvelous Designer was used for cloth simulation to create the base for specific parts of the model such as the material wrapped around the main character's body (Tamnev, 2016, para. 7), and the destruction of the main character's mask was created with the Bullet physics plugin for 3ds Max (Tamnev, 2016, para. 17). The clothes and accessories' animations were

[12] Translation of the vertex information of a 3D model into a bi-dimensional (U and V axis) plane. The resulting coordinates are used to define how a 2D image texture is applied in the 3D model (Pluralsight, 2014)

created by layering baked dynamics simulations[13] on top of the motion capture animation of the character, while some of the simulations such as the movement of the cables holding Adam were translated into a hierarchy of bones and imported into Unity as a FBX file. Some of the characters' models and clothes simulations are available in the Unity Asset Store to be downloaded for free (Pavlov, 2017, para. 31).

CaronteFX is the rigid and soft body[14] dynamic solver in Nextlimit's widely used Real Flow software, among others. The software was adapted to work with Unity as a plugin[15] by reducing the accuracy of the simulations (3D World, 2016, p. 22) and using low-resolution geometry to speed up the calculations, which was then translated into the high-resolution models (Pavlov, 2017, para. 26). CaronteFX ended up being used in almost every shot of the film to simulate movement of environment assets, object fracture[16], destruction and clothes dynamics[17] (3D World, 2016). The use of realistic – physically based – simulation in real-time filmmaking or VR increases the immersivity of the experience for the final user.

The particle simulations used in Adam were created either by using external software such as PhoenixFD or Unity's native particle system (Pavlov, 2017, para. 3-7).

The main character's rig was a mixture of manually and procedurally animated bones and baked simulated deformations. The main rig was created in 3ds Max and later imported into Motionbuilder to clean and tweak the motion captured data (Nechevski, 2016, paras. 10-13). By using a game engine, a game development approach to animation can be taken by creating multiple versions of an animation loop or transition that can later be assembled and replaced (Nechevski, 2016, para. 22).

Most of the characters' animations were created using motion capture technology (Muller, 2017, para. 18). Unity's Animator Controller system was also used to create multiple versions of the characters in the crowd of convicts leaving the prison, with the higher resolution model going up to 30.000 vertices, and the lower resolution model having only 900 vertices (Tamnev, ADAM - Assets Creation for the Real Time Short Film, 2016, para. 12-13).

Adam accomplished its goal of showing the advancements of technology in real-time rendering engines. In *Adam*, a small team of artists created a short film that uses technologies and techniques such as real-time area lights, physics simulation, volumetric fog, transparency shaders, motion blur, among others, and produced a result that can be played in real-time at a 1440p resolution with an NVIDIA GeForce GTX 980 graphics card (Unity Technologies, 2016, paras. 3-5).

2.6. Chapter Conclusion

3D Animation is a fairly recent technology with a lot of potential, where most techniques still used today were created in the late 20th century and have been improved over time.

[13] Consolidation of the simulation calculations' results into a geometry sequence (Trammell, 2016, paras. 8-11)

[14] Soft bodies and rigid bodies are 3D models that are used in a physics simulation. Soft bodies, unlike rigid bodies, can change shape when affected by physics forces or collisions (Sanders, 2017)

[15] Software that adds new features to a computer program (WebWise Team, 2012)

[16] Technique used by physics engines to divide a 3D model into pieces to simulate breaking or shattering

[17] Technology used to simulate the behaviour of clothing or fabric in 3D models (CLO Virtual Fashion, Inc., 2017)

The animation industry has caused numerous improvements in hardware and software, with its technology being used in multiple fields such as entertainment, architecture and science. The main industries that pushed the boundaries of this technology were the 3D animated film industry, made popular by Pixar's animated films and the use of animation for visual effects in feature films such as in the Star Wars' prequel trilogy and the motion captured Gollum character in Lord of the Rings.

Although only being used as proof of concept by the companies that created them, films such as Marza's The Gift and Unity Technologies' Adam continue to push the boundaries of technology development for film production with real-time rendering, accomplishing great results even when using small teams as it was the case in the production of Adam.

While the technology being used for real-time rendered short films is still not fully developed, having a few disadvantages over the traditional methods, it also has substantial advantages such as giving the artists more freedom and quicker previews, allowing for more experimentation during the production stages of the film. Most of this technology is either already available to the public for free, or will be in the near future.

The main 3d game engines in use today have the capability of being used for 3D short films production, each with their advantages and disadvantages. The choice of engine will depend on project requirements and budget. Real-time rendered graphics have now reached the point where the realism and quality of the results are almost as good as when using the traditional rendering methods, with Physically-Based Rendering (PBR) being one of the main technologies that allowed for these results.

PBR is not only used for photorealism, but also for artistic rendering results where accurate calculations are still useful. While an understanding of general light physics principles such as Energy Conservation and how light interactions are calculated in rendering engines is very important for the creation of realistic or accurate materials and lighting, most of the knowledge required for using PBR is roughly the same as for the traditional methods.

There are now free or inexpensive solutions for the low-budget studios or individual artists that are as capable as commercial software, which allows young artists and independent studios to experiment with 3D animation technology. Despite that, some technologies such as motion capture remain too expensive and inaccessible for low budget production, and the creation of 3D animated films requires specific knowledge of different software and techniques, which makes it hard for one single person to handle all the production steps by themselves.

3. Analysis of the Production Pipeline

3.1. Traditional Pipeline Overview

It is important to note that the traditional production pipeline as stated in this chapter can have multiple variations, even inside the same studio, being adapted from case to case, depending on the project[18].

"The production process is never entirely linear, and many tasks are interdependent, overlap and take place in parallel." (Kerlow, 2004, p. 60)

The workflow in analysis here is one that has been widely used in big studios such as Pixar.

3.1.1. Pre-Production

"The first process in the animation pipeline, and also one of the most important, is pre-production." (Gulati, 2010, para. 4)

The pre-production stage includes non-visual tasks such as screenwriting, casting, planning the management of the project, and visual tasks such as storyboarding and development of the look and mood of the project (Kerlow, 2004, p. 59). In this stage, the artists create the ideas, story and designs, while the management team establishes the production plan. This stage is often the most time consuming, and most of the major decisions about the final look of the film are defined before entering the production stage, such as the camera placement, mood and feeling, music, sound, etc. (Beane, 2012, pp. 22-24).

Pre-production begins with a concept or idea that will be developed into a full story. When an idea is accepted, a document with a summary or synopsis of the story is created and reviewed, often multiple times before the script is written (Vardanega, 2013, paras. 4-6). This document often includes some basic additional details, such as characters' small descriptions or important story moments to be included (Beane, 2012, pp. 25-27).

To finalize the development of the story, a storyboard is developed to help visualize the animation and clearly communicate the ideas. The storyboard is the first visual representation of the entire story, and includes the first ideas of camera placements, visual effects and key character poses (Beane, 2012, p. 30). At this stage, temporary voices are recorded to complement the storyboards, which are then turned into a reel or animatic, which allows the film's pitch to be made without a person explaining the story. This is an essential step to validate the strength of the story itself (Vardanega, 2013, para. 9). The animatic can be as simple as an animation created by giving timing and dialogue to each storyboard image, or it can also include more details, such as rough character movements. The storyboards and animatics can be easily edited and rearranged until a final structure to the story is accomplished, something that would be much more expensive to do with the final 3D animation (Beane, 2012, p. 30).

[18] Vd. annex A – 3D Production Pipeline for a visual representation of one of the most common variations of the traditional 3D production pipeline

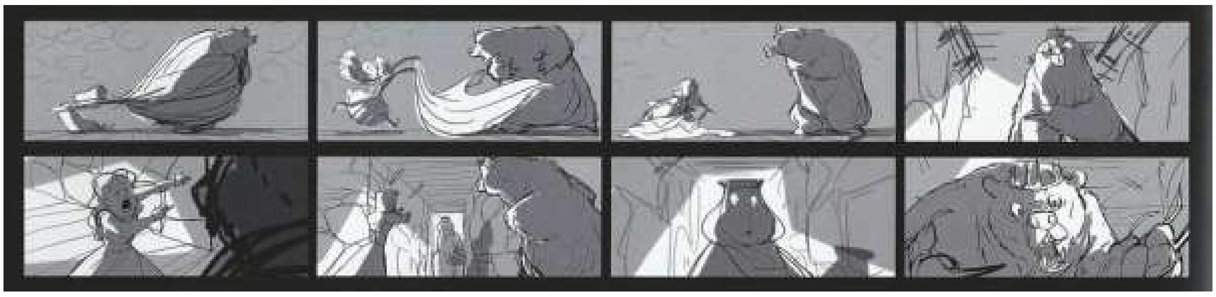


Fig. 14 - Storyboard excerpt from Pixar's Brave (2012) feature film

After the storyboards or animatics are approved, the art department creates the look and feel of the film by illustrating the world, characters and visual appearance (Vardanega, 2013, para. 10). Model sheets are created for the characters, with multiple character expressions and poses, in order to aid in modelling and animation, and keep consistency when a character is being worked on by multiple artists (Gulati, 2010, paras. 9-10).

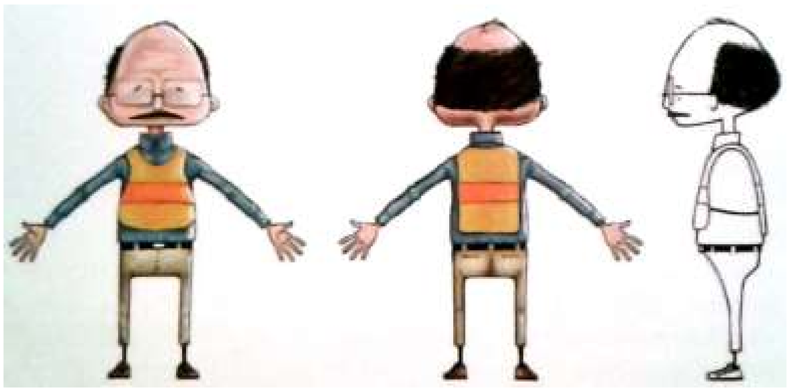


Fig. 15 - Character in the default modelling pose

In some cases, the animatic is only created at the end of the pre-production phase, or another, more finalized one is made, with VFX motion and animation timing, which allow the director to better plan the staging of the sequences and how visual effects are integrated into the final shot (Gulati, 2010, paras. 11-12).

3.1.1. Production

For the production stage to start, all of the pre-production decisions must already be defined, since changes made at the production stage tend to be much more expensive (Beane, 2012, p. 33). During this stage, the Research and Development team is constantly working with all the other artists in order to prevent and solve problems in the pipeline, and allowing for a better overall team workflow by creating scripts and tools for the artists (Beane, 2012, pp. 37-38).

In the layout phase, a 3D version of the animatic is created using simple 3D prototype low-resolution models with roughly the same size and shape than the final models, and some foleys[19] and soundtracks, as well as the final voice-over are then recorded and edited into the layout (Beane, 2012, pp. 37-38). These models use simple geometry pieces such as cylinders and spheres which are parented to the rig's joints, and later replaced by the final high resolution geometry. Another way to create low resolution models, depending on the production pipeline, would be to use software to automatically reduce the final models' geometry resolution. These methods allow for a fluid workflow without too much need for processing power while

[19] Sound effects recorded in post-production for films or games (Mackinnon *apud* Freeman, 2013)

animating (Cantor & Valencia, 2004, p. 267). The following steps of the film production will always be edited on top of this film template, and they can start being worked on at the same time as the layout is being developed (Beane, 2012, pp. 37-38). Other elements such as the cameras' movements, depth of field and scene composition are reviewed and approved by the director (Gulati, 2010, para. 15).

The modelling is usually divided into two departments (organic and non-organic assets) or more, by turning the 2D concepts and model sheets into 3D assets (Gulati, 2010, paras. 16-17). 3D models are created using 3D modelling/animation software such as Autodesk Maya. Other technologies can be used to create 3D models, such as laser scanning real world objects, digital sculpting (which will later need to go through a retopology process) or they can be created procedurally through mathematical algorithms (Beane, 2012, pp. 37-38).

The default material applied to 3D models in most 3D applications is a gray-shaded material without texture. To add an additional layer of interest to a short film, materials are created and applied to the models to provide them with attributes such as color, reflectivity and translucency (Cantor & Valencia, 2004, p. 271).

"If you are going for an abstract or cartoony look, you would likely simplify or exaggerate the natural color and textural qualities of your wooden objects." (Cantor & Valencia, 2004, p. 272)

Texture creation is followed by rigging, and creation of facial expressions. The rigging department creates the assets' rigs and perform tests and required corrections to the rigs. This department is often also involved in the creation of cloth simulation (Gulati, 2010, paras. 23-24).

The animator starts by using the previously made layout as a starting point to create the animation, which can be created using hand-made keyframes, motion capture technology or procedural animation (Beane, 2012, pp. 40-42). If motion capture techniques are used, the clean-up of the captured motion must be performed. Only after the animation is completed the textures and materials are added to the scene and the final simulations are applied (Vardanega, 2013, para. 16) – this may vary from case to case.

Usually, only after the models and animation are completed, the lighting artist will create the light setup for each scene (Derakhshani, 2015, pp. 2-3).

A Visual Effects artist will create animations or simulations for fur, hair, cloth, fire, water, dust, or any other elements that are required in the environment, usually using a physics engine (Beane, 2012, pp. 40-42). This step is often the second to last production step, since it is computationally heavy and its visual aspect is dependent on the other elements (Gulati, 2010, para. 26).

The last production step is the rendering of the animation into multiple render passes that can be edited together in the post-production stage (Beane, 2012, pp. 40-42). In the rendering process, the models are loaded, camera properties such as depth of field and focal point are set up, the lighting is set up, the characteristics of the models' surfaces such as color, texture and reflectivity are specified, shading techniques are chosen, and then the final rendered image is generated (Kerlow, 2004, p. 160).

3.1.1. Post-Production

Post-production refers to all the tasks performed after the final renders are completed.

In the postproduction stage the renders are edited together and visual effects and corrections are applied to them. Often there are errors in the production stage that will be corrected in postproduction. In this stage compositing software is used to merge and edit all the shots and render passes together (Beane, 2012, pp. 43-45).Techniques such as color grading, rotoscoping, masking and the addition of visual effects into the rendered scenes are used in this step (Gulati, 2010, para. 30).

The sound department creates, edits and assembles the required captured or generated sounds into the film.

The final shots are then selected and rearranged to create a seamless final product, and the titles and credits are then added to the final composition and rendered into a video (Gulati, 2010, para. 32).

3.2. Planning and Methodology

Many techniques can be used to optimize workflow when using Unity and create an efficient pipeline. These techniques need to be planned beforehand, in order to keep track of the assets being created and the methods being used for each one (Robin, 2013, para. 11).

3.2.1. Software Set-Up

Before creating 3D assets with the purpose of being imported into Unity, some settings and practices must be taken into account to avoid problems at a later stage.

File names, object names and hierarchies should be simple and descriptive, and the use of special characters should be avoided (Unity Technologies, Art Asset Best Practice Guide, 2017, para. 4).

Unity uses a metric scale (1 unit equals 1 meter in Unity's virtual world). In different software, the scale may vary – in 3DS Max the default scale is in inches, and in Maya in centimeters – so it is recommended that this parameter is set up in the 3D modelling software beforehand (Unity Technologies, Art Asset Best Practice Guide, 2017, para. 3).

3.2.2. Creating 3D Assets

Dividing the assets into two types – unique and common objects – can help the team decide which objects must be created from scratch, going through all the production steps, such as sculpting, modelling, rigging, textures and maps creation, and objects that can be reused from previous projects or modified from one main object, as it often happens when creating different rocks, trees or other elements that require some variation to increase the scene's realism (Robin, 2013, paras. 4-6).

To create reusable assets in Unity, the use of *prefabs* is recommended. A *prefab* in Unity is a reusable asset that remains stored in Unity's project view. Whenever the user inserts a *prefab* into the game, an instance of the stored object is created and remains linked to the original (Unity Technologies, Prefabs, 2012, para. 1), inheriting any changes applied to the original object at any time throughout every instance of it in the scene, including its properties and attached scripts or other components (Unity Technologies, Prefabs, 2012, para. 6). Copying an object multiple times without using a *prefab* creates duplicates of the object that are independently editable, which means that modifying one of the objects will not modify the others. By using *prefabs*, all the objects are linked to the original *template* object, which will propagate all modifications of the original object to the instanced *prefabs* (Unity Technologies, Prefabs, 2017, paras. 1-2).

Reutilization of models is one way to create different assets with minimal effort. When creating, for instance, a brick for a destroyed wall, that object can be reused as a wooden plank, a shelf or any object with similar geometry, just by making minor changes to it and its textures and maps (Robin, 2013, paras. 7-10).

Meshes should always have an efficient and optimised topology, with as little polygons as possible, taking into account that, when possible the polygons should be evenly spaced across the geometry in order to avoid lighting issues (Unity Technologies, Art Asset Best Practice Guide, 2017, paras. 5-6).

3.2.3. Textures

Textures for Unity should be created with values to a power of two, such as 512x512 or 256x1024, to be more efficient and avoid rescaling by the engine at build time. Despite Unity allowing textures up to a 4096x4096 resolution, the maximum recommended is 2048x2048, since some graphics cards and platforms restrict the textures to this maximum size. The artist should create the textures outside Unity in a higher resolution and then rescale them down if needed – scaling textures up is not recommended, since they will lose quality. Textures that require an alpha channel, should not be mixed with textures that do not. It is more effective if they are separated into different texture files. It is better to use lossless image formats, since Unity automatically handles the compression of the images depending on the output platform (Unity Technologies, Art Asset Best Practice Guide, 2017, paras. 7-9).

3.2.4. Rendering Paths

There are two rendering techniques (paths) Unity can use to render the scenes – Forward or Deferred Rendering. The decision on which path to use should be made at the start of the project so that it can be built taking it into account (Unity Technologies, Choosing a Rendering Path, 2017, para. 1).

When choosing the Rendering Path being used by Unity, only real-time lights should be taken into account. The limitations of each Rendering Path only apply to real-time lights, since the number of baked lights in a scene does not influence the real-time rendering processes (Shankar, 2017, para. 11).

Forward Rendering is the default rendering path in Unity. This path is very fast, uses less computer resources, is compatible with most mobile devices, supports anti-aliasing and translucent materials (Shankar, 2017, para. 4) and allows for the use of custom shading models and hardware techniques such as Multi-Sample Anti-Aliasing (MSAA) (Unity Technologies, Choosing a Rendering Path, 2017, para. 2-4). Forward Rendering is better for low-complexity scenes with a small amount of objects and lights, since the performance decreases as scene complexity increases (Shankar, 2017, para. 8).

This process involves creating, for each object, a different render pass for each light affecting it, which means that this process gets slower for each light added to the scene (Unity Technologies, Choosing a Rendering Path, 2017, para. 2-4).

Deferred Rendering requires more powerful hardware, and it is not supported by some mobile hardware (Unity Technologies, Choosing a Rendering Path, 2017, para. 5-6), but its performance cost is roughly fixed, which means that a lot of lights and geometry can be added to the scene without an impactful decrease in performance, making this option the recommended one when the scene needs a large amount of real-time lights (Shankar, 2017, para. 5).

3.2.5. Lighting and Cameras Set Up

Before starting to set up the lighting in a scene, the color space to be used by the engine should also be chosen. There are two color spaces available in Unity: linear and gamma.

"Color Space determines the maths used by Unity when mixing colors in lighting calculations or reading values from textures. This can have a drastic effect on the realism of your game (...)" (Unity Technologies, Choosing a Color Space, 2017, para. 1).

Linear color space produces realistic results, as the light will linearly brighten the scene as its intensity is increased. Gamma color space is mostly used when the destination hardware does not support the use of linear color space, such as with some mobile hardware and game consoles (Unity Technologies, Choosing a Color Space, 2017, paras. 2-5).

The High Dynamic Range (HDR) is a technique used by Unity's virtual cameras to increase image quality, and it produces the best results when used in combination with Linear color space (Unity Technologies, High Dynamic Range, 2017, paras. 1-2).

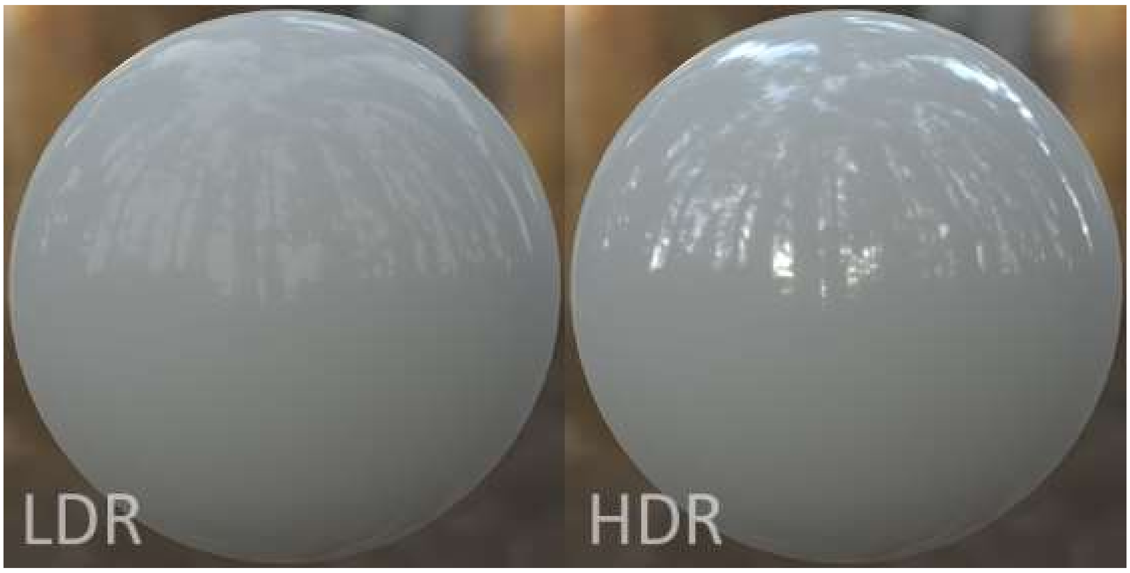


Fig. 16 - Reflections using LDR and HDR environments

The standard camera's dynamic range in Unity is Low Dynamic Range (LDR), which stores colors using 8 bits per channel, giving a total range of 16 million colors. By using HDR, colors are stored with greater precision mimicking real-world lighting (see Fig. 16 – Reflections using LDR and HDR environments) and adding more realism to visible light sources and special effects such as particles (Unity Technologies, High Dynamic Range, 2017, paras. 3-6). This technique can be used in combination with the Tonemapping script (included with Unity's Standard Assets pack), which allows for the control on how the engine handles very bright light values (Unity Technologies, High Dynamic Range, 2017, paras. 7-8).

3.2.6. Animation Sequencing Method

Unity, being a Game Engine, is not primarily made for handling film production. When creating a short film inside Unity, the animations should be treated in the same way as in-game cutscenes, either by using Unity's built-in tools or external plugins.

Plugins such as the Cinema Director – Sequencer & Cutscene Editor can be used within Unity to visually manage cutscene creation and event sequencing. It also allows for the manipulation over time of Unity Game Objects' properties, creation of camera cuts, managing audio tracks, among other features (Cinema Suite Inc., 2017).

Flux is another cinematic editor for Unity. It allows the creation of cutscenes (or sequences) and the management of complex User Interface animations, removing the need for the creation of scripts (Afonso, 2016).

Creating cutscenes in Unity can also be accomplished by multiple methods without the use of any external Plugins, but these methods would be more time-consuming and, in some cases, require specific knowledge of other software.

Since the goal of using Unity for the creation of a short film is to use the Engine for the actual rendering of the scene, using pre-rendered video, albeit being a possible solution (Theriault, 2014, para. 2), would have no benefits over traditional rendering methods.

Creating in-game sequences within Unity without external plugins requires the creation of scripts and event handling, which are tasks that hinder the creative process (Theriault, 2014, paras. 3-4), undermining one of the main advantages of using Unity as a rendering engine for short film production.

Unity's Animation Sequencer, while still not available, will be included in one of the next Unity releases. This tool will allow for the same functionality of the previously mentioned external plugins[20].

3.2.7. Other Optimization Methods

In Unity, objects can be marked as *static* when they are immobile and do not require Unity to consider if they are going to perform any type of movement during gameplay. Knowing that the object will not be moving, Unity can often pre-compute some of the object's required calculations, which reduces the amount of calculations required during runtime. The user can define which of these calculations (lightmaps, reflections, ambient occlusion, etc.) Unity can pre-compute for each object individually (Unity Technologies, Static GameObjects, 2017, paras. 1-2).

3.3. Modelling

The modelling workflow for traditional or real-time rendering can be roughly the same. Both techniques require a certain amount of optimization in the models, since more geometry will result in longer render times.

[20] Vd. chapter 3.5 – Rigging and Animation for more information on the Animation Sequencer

When using a game engine to create an actual game, an estimated maximum polygon count is usually defined for each type of asset depending on the distance from the camera or the importance of the asset itself, and the number of assets in the scene at one time. Going over the polygon count limit is likely to affect the gameplay of the final product, by slowing its response time or increasing the loading times significantly.

Creating a short film using a game engine does not require that it can be played in real-time. The main goal of using a game engine to create short films is to be able to preview the near-final product as soon as possible throughout the production stage. Even if the engine is slowed down by an excess of geometry or textures resolution, as long as it can produce a final rendered image in a short amount of time and the editor software itself can handle the amount of geometry in the scene, the advantages of using it still apply.

"We no longer have to spend hours rendering frames only to find we need to adjust a character's eye highlight. (...) As a result, we save a great deal of time in our production." (Nom, N. *apud* Failes, 2016, para. 7)

Nevertheless, models with a very high resolution such as sculpted models imported directly from ZBrush must be put through a retopology process in both rendering workflows, not only to lower render times, but also to avoid slowing down the other software in the pipeline.

In both workflows, the first versions of the models are often the proxy models, created with roughly the same proportions of the final objects, and used in 3D layout. These models will be gradually detailed and refined throughout the production stage (Tamnev, ADAM - Assets Creation for the Real Time Short Film, 2016, para. 3).

Sculpting software such as ZBrush is used to create some of the high-resolution models, such as characters or organic assets. A lower resolution model is then created with the higher resolution model as a reference and Normal, Occlusion, Curvature and other maps are then generated from the higher resolution model[21] (Tamnev, ADAM - Assets Creation for the Real Time Short Film, 2016, paras. 8-9).

In order to lower rendering times and overall processing from the software, multiple Levels of Detail (LOD) can be created for some models[21] (see Fig. 17 – Different Levels of Detail on the same object).

[21] Vd. chapter 2.5.2 – Adam (2016) – Unity Technologies for more detailed information on each of the methods used by the production team of this real-time rendered short film

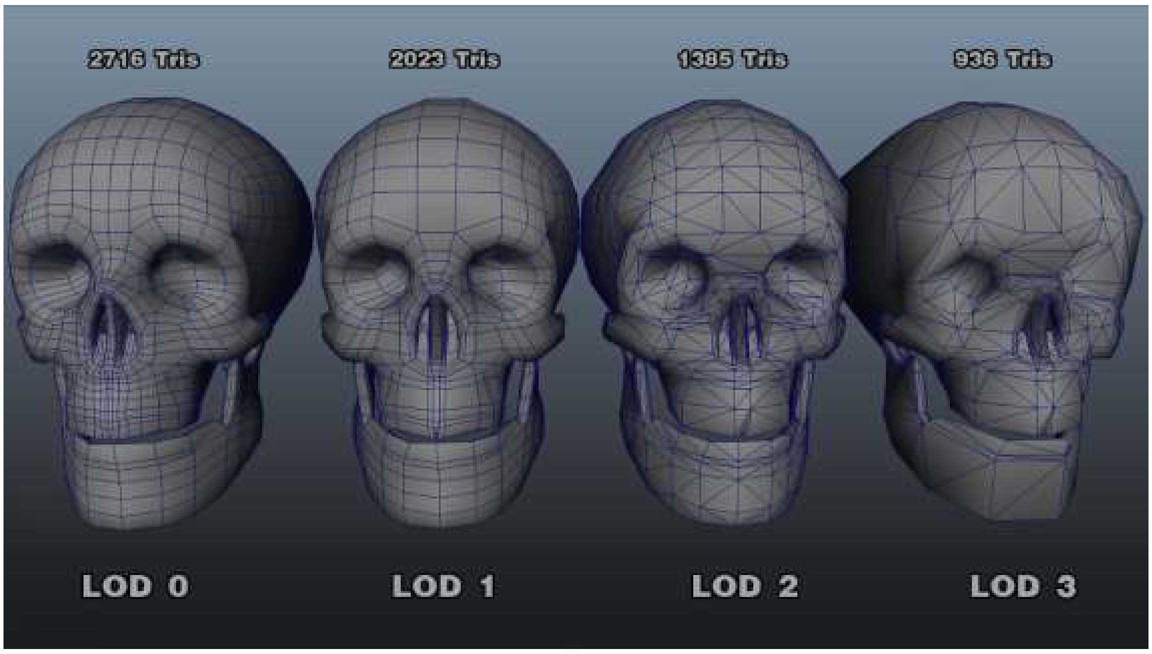


Fig. 17 - Different Levels of Detail on the same object

3.4. Procedural Asset Creation and Animation

Procedural techniques can be used inside or outside of Unity’s editor. Assets and animations can be created procedurally with external software such as SideFX Houdini and exported as an object or sequence of objects[22], or mathematical functions can be used with Unity’s programming capabilities to procedurally create object movements or generate objects (Calderón, 2012, p. 17).

Procedural animation can also be used to create restrictions or constraints for certain movements on complex rigs. If the final result is to be played in real-time, as in a game or interactive application, procedural animation can sometimes require too much time to calculate next frame’s objects’ positions (Calderón, 2012, p. 18).

Programming in Unity can be accomplished by using one of the supported programming languages: C#, Boo and UnityScript, which is similar to JavaScript (aleksandr, 2014, para. 1). Although Boo scripts created in previous versions of Unity will still work, this language’s support was dropped since the version 5.0 of Unity, and the Unity team began focusing on promoting the use of C#, since it was the most used language, and currently most tutorials and sample scripts created by Unity’s team are written in C# (aleksandr, 2014, para. 2-6).

Similarly to the *Mesh* class for procedural mesh generation, textures can also be generated procedurally in Unity, using the *Texture* and *ProceduralTexture* classes for modifying textures’ parameters at runtime (Unity Technologies, 2017). Plugins such as NumberFlow by Catlike Coding can also be used to create textures procedurally in the Unity editor. This plugin uses node-based visual scripting which allows for the creation of limitless personalized textures and texture animations, requiring very little storage space. The generated textures can also be exported from the editor (Catlike Coding, 2017).

[22] Vd. chapter 3.4.1 – Physics and Simulation for more information on how to create and import generated simulations into Unity

Unity does not have modelling capabilities, and external modelling software such as 3ds Max or Maya is used to create the assets that are later imported into Unity[23]. Even though Unity does not provide a graphical interface for asset creation, 3D meshes can be procedurally created and their components modified at runtime using scripts (Unity Technologies, Procedural Mesh Geometry, 2017, para. 1).

To access the properties of a 3D mesh, such as its vertices, normals and UV coordinates at runtime, the *Mesh* class must be used. The vertices of a mesh are stored as an array of three-dimensional vectors, where each one stores a point in the object's local space with X, Y and Z axis values, and each of these values can be edited individually (Unity Technologies, Using the Mesh Class, 2017, para. 3).

Using Houdini Engine's Unity Plugin, Houdini's networks of nodes can be imported into Unity with previously defined procedural controls that can be used to manipulate some of the asset's parameters directly inside the Unity editor. While using the Unity editor, Houdini Engine is working in parallel with Unity to generate the assets according to their controls' values. When the Unity project is published, these values are used to bake the final models, so that the Houdini Engine is not required to be installed on the target system (Side Effects Software Inc., 2017, paras. 1-2).

This system allows, among other features, for the propagation of assets along a curve or a surface and for the creation of assets' variations such as adding floors to a house, defining the number of steps in a staircase, changing a model's dimensions without stretching the textures, and choosing between multiple parts of a model included with the Houdini asset, all just by changing the controller's parameters' values (Side Effects Software Inc., 2017, paras. 9-13).

3.4.1. Physics and Simulation

In both traditional and real-time workflows, external software can be used for simulation purposes[24]. When creating simulations for games or interactive applications, physics simulation baking is used often to create realistic physics-based movements in objects where the player is not supposed to interact with the animation – the simulation is baked offline and played in real-time to save processing power – such as some mechanical objects and natural phenomena (water, fire, etc.) (Calderón, 2012, p. 19). In real-time rendering for short films, although this limitation does not apply, since the final result is not meant to have real-time user interaction, the technique is still useful to reduce render times and required processing power.

Importing the simulation results into Unity is not a trivial task. The standard Unity editor does not have a tool that allows the artist to import a physics simulation directly into Unity in a simple and effective manner, but tools such as the Alembic Importer available on GitHub provides the functionality of importing and exporting the alembic file format within Unity (i-saint, 2017).

The method used in short films production, such as in *Sonder* (in production) and *The Gift* (2016) was exporting the simulation results as an Alembic cache file which can be imported

[23] Vd. chapter 3.3 – Modelling for more information on asset modelling for Unity

[24] Vd. chapter 2.5.2 – Adam (2016) – Unity Technologies for more information on the use of external software for the creation of physics simulation

into Unity by using an Alembic Import plugin (Failes, 2016, para. 6) (Milligan, 2016, para. 6), such as the one created by MARZA, which is available on GitHub for download[25].

3.5. Rigging and Animation

In small teams, as was the case in the short film *Adam* (2016), the animator will often be in charge of multiple stages of production[26], such as the animations layout, rigging, character and environment animation, camera animation and motion capture (Nechevski, 2016, para. 4).

As in traditional animation, the first step is building a placeholder model with low-resolution geometry with roughly the same basic proportions of the final model and set up the rig in order to perform animation tests to tweak the rig behaviour and correct errors as soon as possible (Nechevski, 2016, paras. 5-6).

By using real-time rendering, as soon as the first animations are created they can be pre-visualized in Unity with base lighting and materials, which allows for the animator and the rest of the team to adopt an iterative workflow and have a better preview of the final product throughout the production stage (Nechevski, 2016, para. 9).

An Animator Controller in Unity (see Fig. 18 – Unity's Animator Controller) is a state-machine tool with a flow-chart interface that can be used for handling multiple animations (states) within an object, switching between them depending on imposed conditions such as key pressing or simply according to a specified time limit (Unity Technologies, The Animator Controller Asset, 2017, paras. 1-4). This tool can be used to iterate between multiple animations, or multiple versions of the same animation, to create variation when using the same set of animations as it happens when using crowd simulations.

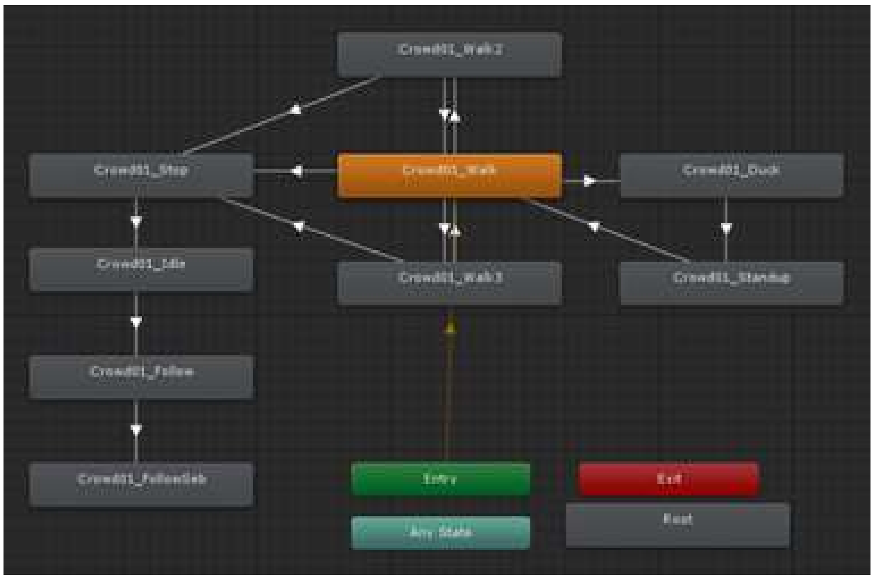


Fig. 18 - Unity's Animator Controller

The Animation Controller is a part of Unity's Mecanim animation system, which also provides retargeting[27] of animations, creation of variables and conditions for the iteration

[25] Vd. chapter 2.5.1 – The Gift (2016) – Marza Animation Planet Inc. for more information about the Alembic Import method created by MARZA

[26] Vd. chapter 2.5.2 – Adam (2016) – Unity Technologies for more information on the rigging and animation methods used in the development of this short film

[27] Reusing animation between different models

between different animation clips, generation of transitions between two animation clips by interpolation, among other useful resources for optimization of the production pipeline[28] (Unity Technologies, 2014).

Unity's Sequencing Tool works similarly to other sequencing tools used in the film industry (Nechevski, 2016, para. 34), such as Adobe Premiere or Apple Final Cut. Although this tool is not yet available to the public, it will be incorporated in one of the upcoming Unity releases. The team that developed *Adam* (2016) used an early prototype of this tool, which already had some additional functionalities such as recording changes in any object's parameters (camera movements, properties and fades), enable or disable objects and change lighting and cameras (Nechevski, 2016, paras. 36-37).

3.6. Materials and Textures

Texturing for real-time rendering engines can be done using the same software used in traditional rendering[29], such as Surface Painter 2, used in the development of *Adam* (2016)[28]. The main difference in the workflow is in the type of shaders used.

Real-time rendering engines often rely on the capabilities of the Physically Based Rendering shaders when pursuing photorealistic results, due to its efficiency. This technology interprets textures in a different manner of most traditional rendering technologies used[30].

As happens for geometry resolution, textures should also be optimized to reduce render times and calculations that may slow down the software being used. Some techniques used in *Adam* (2016)[28] for the models that were farther away from the camera was the use of less material IDs for each asset, and lower texture resolution (Tamnev, 2016, para. 12-13).

3.6.1. PBR Workflow

The most common PBR workflows are the Metal/Roughness and the Specular/Glossiness workflows[31].

The Metal/Roughness workflow is the most widely adopted, since it is less prone to errors and uses 2 grayscale maps[32], which leads to less memory used. There are 3 maps specific to this workflow: base color, metallic and roughness (McDermott, Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2, 2015, p. 3).

The base color map is an RGB map that defines the reflectance value (for metals) or color of the material (for dielectric materials). Since directional light will look incorrect in certain light conditions, this map should not include any light or ambient occlusion information. The reflectance values for metals should be obtained from real-world measured values. Finding accurate and consistent material values for PBR can be a challenge. While some online material libraries like the Quixel Megascans service take measurements from real-world materials, most use values measured in laboratory conditions, which disregard factors like the age, oxidization and wear of the material (Wilson, 2015, paras. 49-50).

[28] Vd. chapter 2.5.2 – Adam (2016) – Unity Technologies for more information on the techniques and software used in the development of this short film

[29] Vd. chapter 2.4.3 – Textures and Maps for more information

[30] Vd. chapter 3.7.1 – PBR Workflow for more information

[31] Vd. annex B – PBR Workflow for a visual representation of both workflows

[32] The Specular/Glossiness workflow replaces one of the grayscale maps with an RGB map, leading to more memory usage

The metallic map is a grayscale map where black values represent dielectric data and white values represent raw, polished metal. This map is used as a mask to differentiate between metal and dielectric data in the base color map (McDermott, Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2, 2015, p. 4). When creating maps that will have an influence on the object's reflectivity, such as metallic or specular (in the Specular/Glossiness workflow) maps, all values refer to how reflective a surface is when looking at it from a zero-degree angle (when the surface is perpendicular to the direction of the camera). Fresnel will then automatically define how reflective the surface is at grazing angles (Wilson, 2015, paras. 28, 37-40).

The roughness map is a grayscale map that describes the surface irregularities (at a microscopic level), where black represents a smooth surface and white represents a rough surface. The roughness of the material changes the direction of the light rays, making its specular reflections more or less focused (McDermott, Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2, 2015, p. 10). Surface variation should generally be represented in this map instead of reflectivity maps, since reflectivity seldom varies in the same material (Wilson, 2015, para. 29).

The maps specific to the Specular/Glossiness workflow are diffuse, specular and glossiness. In this workflow, texture artifacts are less apparent, and the artist has more control over the reflectance values of dielectric materials. On the other hand, depending on the software used, the law of Energy Conservation can be broken, so it requires the artist to take that into account. As McDermott (2015, p. 17) summarized, *"Because the specular map provides control over dielectric F0*[33]*, it can be more susceptible to incorrect values being used. It is possible to break the laws of conservation if it is not handled correctly in the shader."*

The diffuse map is an RGB map that defines the surface color, but not the reflectance value as in the Metallic/Roughness workflow. In this map, the areas that represent raw metal should be black, because metal doesn't have a diffuse color.

The specular map is an RGB map that defines the reflectance values of the material. For dielectric materials, the map should use grayscale values, and for metals it can be colored.

The glossiness map is a grayscale map, similar to the roughness map in the Metallic/Roughness workflow, but with the black/white values inverted (McDermott, Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2, 2015, p. 16).

The maps common to both workflows are optional maps that can be added to increase the realism of the object: the Ambient Occlusion (AO) map defines the quantity of environment light that can reach the surface; the Height map is used for displacement and/or parallax mapping, to add apparent depth to the surface; the Normal map simulates surface detail, which is often also represented in the roughness or glossiness maps (McDermott, Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2, 2015, pp. 18-21).

When creating textures, tweaking the material values to look better in a specific lighting setup is an error. By using real-world material values, the process will be simplified, the result will be consistent regardless of the lighting environment the asset is being subject to, and the assets can be reused in other projects, even in completely different conditions (Wilson, 2015, para. 56).

[33] F0 is the amount of light reflected when looking straight at an object – the surface is perpendicular to camera direction

3.7. Lighting and Light Baking

3.7.1. Light Types

Light types in Unity are similar to what may be found in most 3D software packages. Directional lights, point lights, spotlights and area lights all behave as expected from a 3D artist's experience. Area lights are computationally expensive, even though they can only be used as baked lights, but produce realistic results. Directional lights in Unity 5 have the advantage of being able to control the skybox by rotating this "Sun" light, which will produce the appearance of daylight, sunset, sunrise or night-time depending on the "Sun" light's rotation (Unity Technologies, Light Types, 2017).

Emissive lighting (emissive materials) make objects appear bright and create a soft lighting effect on the surrounding geometry (Shankar, 2017, para. 24). They emit light in a similar way to area lights, and may be a good alternative when using Precomputed Realtime GI, which does not support area lights (Unity Technologies, Emissive Materials, 2017, para. 1). These materials' light only affects objects marked as *static*. In cases where dynamic objects need to be affected, Light Probes must be used in order to achieve that result (Unity Technologies, Emissive Materials, 2017, para. 5).

Unlike area lights, emissive materials can be applied to geometry of any shape (Shankar, 2017, para. 24) and its parameters, such as intensity or color, can be changed during gameplay (Unity Technologies, Emissive Materials, 2017, para. 1).

Ambient lighting is a global light source that illuminates all objects in the scene from every direction. This type of lighting requires a very small amount of resources to calculate (Unity Technologies, Ambient Lighting, 2017, paras.1-4), but unless Global Illumination is being used at the same time, it will create even lighting throughout the whole scene, which can make it appear flat and unrealistic (Shankar, 2017, para. 32).

3.7.2. Global Illumination and Light Baking

Global Illumination (GI) is a computational expensive technology. Game engines often use, as much as possible, techniques to allow for the calculation of GI before the game starts, to reduce calculations during runtime (Unity Technologies, Introduction to Lighting and Rendering, 2017, para. 1).

Unity combines three categories of lighting to accomplish the best results as efficiently as possible.

Real-time lighting is the default lighting scheme in Unity, and its influence on the scene is calculated every frame, during runtime. It is most used when the lights or the objects they are illuminating are moving (Unity Technologies, Choosing a Lighting Technique, 2017, paras. 3-5).

Baked GI lighting is calculated before the application starts, and is used to create immobile (static) lights that do not change during gameplay. In this technique, direct and indirect (bounced) light's influence in the scene is calculated and merged into the scene's textures, taking into account the object's materials properties (Unity Technologies, Choosing a Lighting Technique, 2017, paras. 6-9).

Precomputed Realtime GI Lighting is a technique used by Unity that allows the engine to create mobile light maps with GI, making most of the expensive calculations before the application starts, by generating low-resolution approximations (clusters) of the static geometry in the scene (Unity Technologies, Choosing a Lighting Technique, 2017, paras. 10-20).

Using both Baked and Precomputed Realtime GI is not recommended, since the computer will need to use roughly double the time in calculating them both – Unity's team recommends choosing the one that best applies to each project, based on the destination hardware's capabilities, and opting by using only Baked GI in devices with low processing power and/or limited video memory (Unity Technologies, Choosing a Lighting Technique, 2017, paras. 21-23).

These lighting settings can either be controlled on a global level (enabled or disabled for the whole scene) or for each light individually. For hardware with limited processing power it is recommended to disable either Baked or Precomputed Realtime GI lighting globally, to make sure the engine does not use both techniques at the same time, since when using real-time lighting in an individual light, the indirect lighting from that light source may still be processed by Unity's Precomputed Realtime GI system automatically (Unity Technologies, Choosing a Lighting Technique, 2017, paras. 24-28).

Baking lighting information into objects' textures allows for the calculations of that lighting to be done only once (Tino, 2015, para. 1). Baked lighting produces more realistic shadows and higher quality results than real-time lights[34] and also increases performance, hence it should be used as much as possible. Since light baking is not done in real-time, the quality settings can be increased at the time of the bake and later be decreased for the final product real-time lighting (Shankar, 2017, paras. 12-13).

This technique, however, often creates undesired results when not used properly. Using the correct processes in creating the lightmaps that will store the lighting information is a crucial step to get clean results in the final product (Tino, 2015, para. 1).

Unity, be default, uses the object's UV0 channel – which is usually used to store texture information – to bake lighting information. One of the most common mistakes that can happen when using this technique is having an object with overlapping UVs in this channel, which will create undesirable results. This can be avoided by separating the overlapping areas in the main UVs, by creating a different set of UVs in the UV1 channel that will be used by Unity, or by telling Unity to generate the lightmap in the UV1 channel itself (on the object importer settings). The automatic method can produce good results in non-organic objects, but it is not recommended for organic objects, where it will most likely create visible seams in the UV mapping (Tino, 2015, paras. 6-7).

Lightmap resolution is measured in texels (texture pixels) per unit. Increasing this value will create larger maps, but with better quality. (Tino, 2015, para. 9)

3.7.3. Light Probes and Reflection Probes

When using baked lighting, dynamic objects will not be affected by it. To provide dynamic objects with some of the baked lighting information (indirect lighting), Light Probes

[34] Vd. annex C – Before and After Light Baking in Unity for a visual comparison illustrating the advantages of using light baking in Unity 5

must be used (Shankar, 2017, paras. 21-23). Light Probes are points in space that capture light information from all directions, which are later used as a much less computationally expensive method to blend indirect light information with dynamic meshes that would otherwise receive no global illumination from the environment (Unity Technologies, Light Probes, 2017, paras. 1-3).

By default, the reflections on the objects are created by using Unity's skybox textures. When objects are indoors, or somehow occluded from the sky, Reflection Probes may be used to create more realistic reflections on the object. Despite giving more accurate results, these probes affect the performance of the application (Unity Technologies, Reflections, 2017, paras. 4-6).

"These probes render the world from their position in 3D space and write the results to a cubemap. This can then be used by nearby objects to give the impression that they are reflecting the world around them." (Unity Technologies, Reflections, 2017, para. 4).

3.8. Rendering

There are multiple ways of capturing a frame in Unity. Depending on the project's needs, the technique used may vary.

3.8.1. Single-Pass Rendering

If single-pass frame capturing is enough for the project, a simple script can be created for this end, or one of the already existing ones can be used instead.

ScreenCaptureImageSequence.cs is a script for Unity that allows for the capture of image sequences in Unity at a maximum rate of 60 frames per second. This is an example of a script that could be used for rendering Unity's output into a sequence of images (Hyshinara, 2015).

3.8.2. Multi-Pass Rendering

If the project requires multi-pass rendering to allow for more flexibility in post-production in software such as The Foundry's NUKE, the process can be more complex without the use of external plugins or scripts.

In a game engine such as Unity, the rendering passes are not the same as in a traditional rendering engine[35].

As stated by Unity Technologies in Unity's Documentation, *"ForwardBase pass renders ambient, lightmaps, main directional light and not important (vertex/SH) lights at once. ForwardAdd pass is used for any additive per-pixel lights. (...) Deferred pass renders all information needed for lighting (in built-in shaders: diffuse color, specular color, smoothness, world space normal, emission). It also adds lightmaps, reflection probes and ambient lighting into the emission channel."* (Unity Technologies, Unity's Rendering Pipeline, 2017, paras. 4-6).

These render passes differences between traditional and real-time rendering engines are not ideal in the sense that, even if the artists had access to the individual passes created by

[35] Vd. chapter 2.4.4 – Lighting and Rendering for more information on traditional rendering engines' render passes

Unity's Forward or Deferred rendering paths, the post-production workflow would change significantly, and the passes created would not allow for the same control over the final product that the traditional rendering engines' passes provide.

Plugins such as Offline Render for Unity allow for the rendering of the scenes as multi-channel EXR images in real-time, including the ability to render separate channels for depth, per-light shadows, diffuse, ambient occlusion, specular, reflections, object ID pass, among others to be used in a post-production software such as NUKE similarly to traditional rendering methods. This plugin also allows the project developers to use its API to create their own custom passes (You can do it! VFX, 2017).

3.9. Sound and Post-Production

3.9.1. Sound

Sound for a real-time rendered short film is not created in the same way as sound for a game or interactive application. Interactivity implies that the sounds must be played as a reaction to a user's action, hence they must be created directly in the game engine.

When creating sound for a real-time rendered short film, the process used is, in great part, the same as creating sound for a film rendered with the traditional method. Instead of creating the sound in the game engine, it will be created and synchronized over a rendered video file. The main difference being that when using real-time rendering, as a result from the iterative nature of its workflow, more versions of the scenes will be created and sonorized. As stated by Aleksander Kashikoff, sound designer in the *Adam* (2016) short film, *"Although it is rendered real-time, the sound was made against a rendered file, more or less as if it was a normal film – with the difference that character action, camera angles and movements, textures and fabrics were subject to change at any given moment (...)" (Andersen, 2016, para. 9).*

3.9.2. Image Effects and Shaders

Image Effects, in Unity, are scripts that can be attached to a camera object inside a Unity scene to modify its render output. Although these are standard C# programming scripts, the computation of the results is done by using shaders[36]. The type of shaders most used for post-production effects in Unity are called *screen shaders* (Zucconi, Screen shaders and image effects in Unity3D, 2015).

By controlling the way a shader interprets the information on the scene, the developer can create visual results such as *vignette*, chromatic aberration, image displacement, or other cinematic or creative effects (Zucconi, Screen shaders and image effects in Unity3D, 2015).

Unity provides for free a Post Processing Stack asset that combines multiple image effects, such as antialiasing, ambient occlusion, fog, depth of field, motion blur, bloom, color grading, among other effects (Unity Essentials, 2017), and other post-processing packages are available for free or at a cost, in the Unity Asset Store, such as the Camera Filter Pack that merges more than 300 effects into a single package (Vetasoft, 2017) or the Amplify tools, that

[36] Program specifically created to run on a GPU (Zucconi, A gentle introduction to shaders in Unity3D, 2015)

feature texuring, color grading, LUT packs[37], node-based shader creation, among other effects (Amplify Creations, n.d.).

3.9.3. Lookup Textures (LUTs)

"[LUTs] can be created in Photoshop to add tints & color correction, modify shadows, blow out areas, and apply the sort of Photoshop image corrections to each frame of your scene." (Shankar, 2017, para. 33)

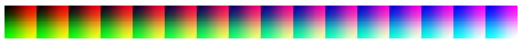


Fig. 19 - Lookup texture in Unity

Lookup Textures (LUT) Color Correction is a technique that can be used in Unity to perform color grading in a scene. In this technique, the artist uses external software, such as Photoshop or Gimp, to edit a single specific texture (see Fig. 19 – Lookup texture in Unity) that Unity will use to automatically perform the color grading (Unity Technologies, Color Correction Lookup Texture, 2017, paras. 1-2).

"Advantages include better performance and more professional workflow opportunities (...)" (Unity Technologies, Color Correction Lookup Texture, 2017, para. 2).

To create a custom LUT, the artist should take a screenshot of the scene, import it into a graphics editor, apply the desired color adjustments, and apply the exact same adjustments to the default (neutral) LUT provided with Unity's Standard Assets Package, which is then applied as the new LUT (Unity Technologies, Color Correction Lookup Texture, 2017, para. 8).

3.10. Chapter Conclusion

Most of the production steps, techniques and technologies used in the traditional rendering pipeline are still used in real-time rendering. A large part of them are created in external software and later exported into Unity for the scene set up and rendering.

While the need for assets' optimization for both rendering methods still exists, it has decreased over time with the development of more efficient and powerful software and hardware. Despite that need, the possibility of almost immediate pre-visualization of the final output allows the artists to promptly evaluate the final results of the product they are creating. One crucial point that differs in the use of game engines for film production or interactive applications and games is that, in film production, the final results are not required to be output in real-time. Even if the rendering cannot be done at the frame rate required for film production (usually 24 frames per second), as long as the rendering of each frame can be done fairly quickly, the advantages of using the game engine still apply.

Working with Unity requires some additional steps on the pre-production stage, mainly involving production decisions about what methodologies and techniques to use that should be specified before the production stage begins, engine and scene set up, and external plugins or the need for the creation of scripts to aid in the development process. The whole production stage should be meticulously planned when working with Unity, since this software was not originally created for film production, and many problems may arise when not properly considered in pre-production.

Procedural modelling and animation techniques are widely used in the traditional rendering pipeline for the creation of variation in models or animations, or simulations of fluids,

[37] Vd. chapter 3.9.3 – Lookup Textues (LUTs) for more information

fire, smoke, etc. These techniques can also be used when working with Unity, although some additional steps will be required, such as using external plugins or scripts to import the assets into the engine.

Although some of the same software can be used for the creation of textures for Unity, the way in which they are used may differ in some points, and knowledge of the Physically Based Rendering Workflow will be required when using PBR shaders in Unity.

The base lighting techniques used in Unity are roughly the same as in traditional rendering methods, since the lights work in very similar ways in both methods. Despite that, knowledge of some Unity specific techniques such as light baking and light mapping, light and reflection probes, and the different Global Illumination methods used by Unity may be required to reach good results while minimizing the render times.

The rendering process in Unity is very different from traditional rendering engines. To reach the same capabilities, such as multi-pass, High Dynamic Range rendering in OpenEXR file format, often easily accomplished in traditional rendering engines, Unity requires the use of external plugins and/or scripts to provide these results. Despite that, using techniques such as Lookup Textures (LUT) and camera effects in Unity, along with the ability to render the scenes in real time, may reduce or eliminate the need for post-production, depending on the project's needs.

Some of Unity's tools, such as the Animator Controller, the automatic management of multiple Levels of Detail for 3D models, and the Sequencing Tool can help artists create efficient workflows and compensate for some of the disadvantages of using a game engine for film production.

4. Final Conclusions and Future Work Perspectives

4.1. Comparison Summary

The main advantage of using a game engine for film production is the real-time rendering capability of the engine, which allows artists to preview the rendered scene almost instantly with a visual quality very close to the final product, giving them more artistic freedom and allowing for more experimentation during the production stage.

Unity is still not completely optimized for the creation of animated films, with some of the production stages lacking tools to achieve the same results as traditional rendering methods, the most obvious being the lack of an importer for physics simulations created in external software and a sequencer tool in Unity's default installation, which would force the developers to install third-party plugins or create their own scripts. Despite that, these faults can be overcome with the use of external plugins and scripts, with the most popular options being free or inexpensive.

While most of the knowledge needed for the creation of short films in Unity is the same as for using traditional rendering methods, there are some steps where specific knowledge of the software or techniques is required, which may be seen as a crucial disadvantage when choosing between the two rendering methods, or as an opportunity to explore different working methods using Unity's tools.

Using a real-time rendering engine does not necessarily mean that the final product will have actual real-time capabilities. If the film is intended to be visualized in a movie format, the final product does not need to be rendered in real-time. On the other hand, if the film is intended to be visualized by the public while being rendered in real-time, many advantages can come from using this approach, such as the ability to update the film after its release, or the inclusion of randomly generated events or animations, which could create different result every time the film is visualized.

4.2. Final Product Analysis

By using Unity and creating a few specific tools to fulfil their needs, Marza accomplished the rendering of huge amounts of geometry at the same time by importing geometry into Unity in an Alembic format and the capture of OpenEXR images from Unity.

Unity Technologies used a number of features such as Real-Time Area Lights, Volumetric lighting, alembic import, and their upcoming Animation Sequencer tool to show that, even at this early stage, Unity is capable of using technically advanced methods to produce results with impressive visual quality, without compromising the real-time rendering advantages of the engine.

Even if we do not take into account that these films were rendered in real-time, their visual quality is impressive. While they may not be able to compete with the larger studio productions' outstanding visual quality – as it often happens in Pixar's short films –, we must not forget that both these films were created by small teams, using software that is still not fully developed or optimized for short film creation, which will, in the near future, suffer many improvements in this field.

4.3. Is Real-Time Rendering a Viable Option?

Marza's and Unity Technologies' teams found methods of dealing with the lack of tools directed for short film production in Unity by developing new tools for their needs. That, along with the analysis of the final results they ultimately accomplished, shows that Unity can be used for the production of short films.

Adam's main goal was to prove that Unity could be used for film production, even when using small teams, and in that aspect, it was a success.

Even though *The Gift*'s team had to spend additional time developing some tools they deemed necessary for the production of their short films, by using Unity they reduced their production time by 20-30%.

With this, we can conclude that, while Unity is still not at an ideal state for short film production, it has been proven that it can, effectively, be used for it. With the following releases of Unity and the addition of tools used specifically for short film production, most of the disadvantages it now presents will be overcome, and it will undoubtedly continue being used for the production of short films.

4.4. The Future of Animated Cinema

Merging different technologies and fields is one way of creating new technology or products that often produces innovation. It allows professionals of different fields to learn from each other and appeals to the creativity of artists and developers, particularly in technological fields, generating interest in the most recent technologies and in finding new ways to put them to use.

By using game engines to create animated films, a vast range of new options are added to the film production industry, such as the creation of interactive cinema, where the users become an active actor in the film, instead of just a passive observer.

New technologies that can be used for that end are already developed and commercially available, such as VR headsets that would create an immersive user experience for an interactive animated film, or AR technology, that could be used to create the illusion of the film characters interacting with the real world.

References

Chu, H.-K., & Lee, T.-Y. (2009). *Multiresolution Mean Shift Clustering Algorithm for Shape Interpolation.* IEEE Computer Society.

3D World. (2016, November). Real-Time Filmmaking. *3D World*, pp. 19-24.

Adobe Systems Incorporated. (2017, February 15). *3D Painting*. Retrieved April 18, 2017, from Adobe Support: https://helpx.adobe.com/photoshop/using/3d-painting-photoshop.html

Adobe Systems Incorporated. (2017, February 15). *3D Texture Editing*. Retrieved April 18, 2017, from Adobe Support: https://helpx.adobe.com/photoshop/using/3d-texture-editing.html

Afonso, N. (2016, December 8). *Flux*. Retrieved May 6, 2017, from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/18440

Akasaki, H. (2016, May 20). *キャラクターモデリング　～まずは顔のモデルから*. Retrieved June 22, 2017, from Autodesk Area Japan: http://area.autodesk.jp/column/tutorial/character_arpeggio/01_modeling/

aleksandr. (2014, September 3). *Documentation, Unity Scripting Languages and You*. Retrieved April 26, 2017, from Unity Blogs: https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/

Amplify Creations. (n.d.). *Unity3d*. Retrieved June 26, 2017, from Amplify Creations: http://amplify.pt/unity/

Andersen, A. (2016, July 4). *The Story Behind the Sound for Unity's Astonishing Adam Demo*. Retrieved April 21, 2017, from A Sound Effect: https://www.asoundeffect.com/unity-adam-sound-design/

aneesharai22. (2016, January 28). *R&D Blog - Week 2 - Part 2*. Retrieved June 21, 2017, from Animationblog: https://animationblog.wordpress.com/2016/01/28/r-d-blog-week-2-part-2/

Atwood, J. (2008, March 10). *Real-Time Raytracing*. Retrieved June 21, 2017, from Coding Horror: https://blog.codinghorror.com/real-time-raytracing/

Bagard, A. (2016, July 14). *Texturing Unity's Adam With Substance Painter*. Retrieved April 20, 2017, from Allegorithmic: https://www.allegorithmic.com/blog/texturing-unitys-adam-substance-painter

Bao, H., & Hua, W. (2011). *Real-Time Graphics Rendering Engine.* New York: Springer.

Beane, A. (2012). *3D Animation Essentials.* Indianopolis, Indiana: John Wiley & Sons, Inc.

Bettinger, B. (2017, January 28). *Pixar by the Numbers - From 'Toy Story' to 'Monsters University'*. Retrieved January 28, 2017, from Collider: http://collider.com/pixar-numbers-monsters-university/

Calderón, R. M. (2012). *Diseño e Implementación de un Algoritmo de Animación Dinámica de Personajes (Máster Oficial en Informática Gráfica, Juegos y Realidad Virtual).* Madrid, Spain: Universidad Rey Juan Carlos.

Cantor, J., & Valencia, P. (2004). *Inspired 3D Short Film Production.* Boston, MA: Thomson Course Technology PTR.

Catlike Coding. (2017, April 11). *NumberFlow*. Retrieved April 26, 2017, from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/12902

Chang, A. (n.d.). *The Process of 3D Animation*. Retrieved May 5, 2017, from Media Freaks: http://media-freaks.com/the-process-of-3d-animation/

Chopine, A. (2011). *3D Art Essentials: The Fundamentals of 3D Modeling, Texturing, and Animation.* Oxford, UK: Elsevier Inc.

Cinema Suite Inc. (2017, April 6). *Cinema Director - Sequencer & Cutscene Editor*. Retrieved May 6, 2017, from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/19779

CLO Virtual Fashion, Inc. (2017). *FAQ*. Retrieved June 26, 2017, from Marvelous Designer: https://www.marvelousdesigner.com/support/faq

Creative Bloq Staff. (2014, May 21). *9 Things 3D Artists Need to Know About NUKE*. Retrieved April 18, 2017, from Creative Bloq: http://www.creativebloq.com/audiovisual/nuke-51411738

Creative Bloq Staff. (2015, February 20). *How Real-Time Rendering Will Change the Way You Work Forever*. Retrieved November 18, 2016, from CreativeBloq: http://www.creativebloq.com/3d/real-time-rendering-21514214

Derakhshani, D. (2015). *Introducing Autodesk Maya 2016.* Sybex.

Digital Synopsis. (n.d.). *46 Famous Movie Scenes Before And After Special Effects*. Retrieved June 21, 2017, from Digital Synopsis: https://digitalsynopsis.com/design/movies-before-after-green-screen-cgi/

Disney. (2012). Retrieved November 18, 2016, from Disney Research: https://www.disneyresearch.com/

Efremov, V. (Director). (2016). *Adam* [Motion Picture].

Epic Games. (2017). *Introduction to Blueprints*. Retrieved January 28, 2017, from Unreal Engine 4 Documentation: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/GettingStarted/index.html

Epic Games, Inc. (2017). *Blueprints Visual Scripting*. Retrieved June 21, 2017, from Unreal Engine 4 Documentation: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/

Failes, I. (2016, December 8). *The Animated Short 'Sonder' Chose an Unconventional Rendering Solution: Unity*. Retrieved April 20, 2017, from Cartoon Brew: http://www.cartoonbrew.com/shorts/animated-short-sonder-chose-unconventional-rendering-solution-unity-145944.html

Foley, J. v. (1996). *Computer Graphics, Principles and Practice* (2nd ed.). Boston, MA: Addison-Wesley Professional.

Free Software Foundation, Inc. (2017). *Processor and CPU Time*. Retrieved June 9, 2017, from The GNU C Library: https://www.gnu.org/software/libc/manual/html_node/Processor-And-CPU-Time.html#Processor-And-CPU-Time

Freeman, W. (2013, May 27). *Audio Special: Foley for Games*. Retrieved June 26, 2017, from Develop: http://www.develop-online.net/analysis/audio-special-foley-for-games/0117620

Fronczak, T. (2011, May 10). *10 Types of 3D Animation Software Worth Knowing*. Retrieved April 18, 2017, from Animation Career Review: http://www.animationcareerreview.com/articles/10-types-3d-animation-software-worth-knowing

Glick, C. (2011, December 1). *Beginner's Guide to Unity*. Retrieved January 28, 2017, from Digital Tutors: http://www.digitaltutors.com/tutorial/572-Beginners-Guide-to-Unity

Gray, A. (2015). *Introduction to 3D Animation*. Retrieved May 5, 2017, from Animation Arena: http://www.animationarena.com/introduction-to-3d-animation.html

Grove, N. (2016, June 26). *Setting Up Physically-Based Materials and Baked Lighting in Unity 5*. Retrieved June 22, 2017, from The Song of Seven: http://www.thesongofseven.com/dev-blog/2016/6/26/setting-up-pbr-materials-and-baked-lighting-in-unity-5

Gulati, P. (2010, June 9). *Step-by-Step : How to Make an Animated Movie*. Retrieved February 27, 2017, from Envato Tuts+: https://cgi.tutsplus.com/articles/step-by-step-how-to-make-an-animated-movie--cg-3257

Hocking, J. (2015). *Unity in Action: Multiplatform Game Development in C# with Unity 5.* Manning Publications Company.

Huang, K.-S. (n.d.). Retrieved November 18, 2016, from Ke-Sen Huang's Home Page: http://kesen.realtimerendering.com

Hyshinara. (2015, August 18). *ScreenCaptureImageSequence.cs v2 by Hyshinara*. Retrieved May 6, 2017, from PasteBin: https://pastebin.com/0gkxxybb

Imtiaz, K. (2014, March 23). *The Last of Us Real-Time vs Pre-Rendered Models Tease the Graphical Jump For Uncharted On The PS4*. Retrieved January 28, 2017, from Gearnuke: http://gearnuke.com/last-us-cutscene-vs-gameplay-graphics-tease-graphical-jump-expect-uncharted-ps4/

Industrial Light & Magic. (2017). *About OpenEXR*. Retrieved May 6, 2017, from OpenEXR: http://www.openexr.com/about.html

i-saint. (2017, April 5). *Alembic Importer / Exporter*. Retrieved April 6, 2017, from GitHub: https://github.com/unity3d-jp/AlembicImporter

IuGher. (2015). *Nature HDRI*. Retrieved June 21, 2017, from IuGher Texture: http://www.lughertexture.com/spherical-hdri-360/spherical-360-nature-trees-plants-sky-hdri

Kajisa, K. (Director). (2016). *The Gift* [Motion Picture].

Kerlow, I. V. (2004). *The Art of 3D Computer Animation and Effects.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Kumari, S. (2015, May 5). *Modelling & Rigging a Cartoon Parrot in Maya: Part 9*. Retrieved June 21, 2017, from EnvatoTuts+: https://cgi.tutsplus.com/tutorials/modelling-rigging-a-cartoon-parrot-in-maya-part-9--cms-23933

Lobo, R. (2014, October 9). *How Star Wars changed the special effects industry*. Retrieved January 29, 2017, from The New Economy: http://www.theneweconomy.com/home/how-star-wars-changed-the-special-effects-industry

Lucasfilm Ltd., Sony Pictures Imageworks Inc. (2017). *Introduction*. Retrieved May 6, 2017, from Alembic: http://www.alembic.io/

Lyapin, E. (n.d.). *Skull*. Retrieved June 22, 2017, from Evgeny Lyapin 3D Artist: http://evgeny-3d.com/3d-modeling/scull.html

Manasseh & Ephraim Studios. (2014, June 18). *The Best Free Post-Production Software for Filmmakers & VFX Artists*. Retrieved April 18, 2017, from Manasseh & Ephraim Studios: http://www.mestudios.com/best-free-post-production-software-for-filmmakers-vfx-artists/

MarcelMarkov. (2012, November 12). *Zbrush Bust - Render Passes*. Retrieved June 21, 2017, from DeviantArt: http://marcelmarkov.deviantart.com/art/Zbrush-Bust-Render-Passes-337452354

Masters, M. (2014). *Unity, Source 2, Unreal Engine 4, or CryENGINE - Which Game Engine Should I Choose?* Retrieved January 29, 2017, from Digital Tutors Blog: http://blog.digitaltutors.com/unity-udk-cryengine-game-engine-choose/

McDermott, W. (2015). *Light and Matter: Practical Guidelines for Creating PBR Textures, vol. 2.* Retrieved February 3, 2017, from The Comprehensive PBR Guide by Allegorithmic: https://www.allegorithmic.com/system/files/software/download/build/PBR_volume_02_rev05.pdf

McDermott, W. (2015). *Light and Matter: The Theory of Physically-Based Rendering and Shading, vol. 1.* Retrieved February 3, 2017, from The Comprehensive PBR Guide by Allegorithmic: https://www.allegorithmic.com/system/files/software/download/build/PBR_Guide_Vol.1.pdf

Microsoft. (2017). *Face and Vertex Normal Vectors*. Retrieved June 26, 2017, from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/bb324491(VS.85).aspx

Milligan, M. (2016, September 6). *Marza Movie Pipeline Pushes Unity Beyond the Gaming Sphere*. Retrieved February 17, 2017, from Animation Magazine: http://www.animationmagazine.net/features/marza-movie-pipeline-pushes-unity-beyond-the-gaming-sphere/

Muller, M. (2017, January 11). *Unity Blogs*. Retrieved March 13, 2017, from Adam - Step by Step: https://blogs.unity3d.com/2017/01/11/adam-step-by-step/

Nechevski, K. (2016, August 31). *Adam - Animation For The Real-Time Short Film*. Retrieved April 22, 2017, from Unity Blogs: https://blogs.unity3d.com/2016/08/31/adam-animation-for-the-real-time-short-film/

NVIDIA Corporation. (2017). *DirectX 11 Tessellation*. Retrieved June 26, 2017, from NVIDIA: http://www.nvidia.com/object/tessellation.html

Pavlov, Z. (2017, January 4). *Unity Blogs*. Retrieved March 13, 2017, from Adam - VFX in the Real-Time Short Film: https://blogs.unity3d.com/2017/01/04/adam-vfx-in-the-real-time-short-film/

Pharr, M., Jakob, W., & Humphreys, G. (2016). *Physically Based Rendering: From Theory to Implementation* (3rd ed.). Burlington, Massachusetts, United States of America: Morgan Kaufmann.

Plante, C. (2015, March 4). *Why video game engines may power the future of film and architecture*. Retrieved January 30, 2017, from The Verge: http://www.theverge.com/2015/3/4/8150057/unreal-engine-4-epic-games-tim-sweeney-gdc-2015

Pluralsight. (2014, January 14). *Key 3D Rigging Terms to Get You Moving*. Retrieved May 4, 2017, from Pluralsight Blog: https://www.pluralsight.com/blog/film-games/key-rigging-terms-get-moving

Pluralsight. (2014, January 19). *Understanding UVs - Love Them or Hate Them, They're Essential to Know*. Retrieved June 26, 2017, from Pluralsight: https://www.pluralsight.com/blog/film-games/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know

Polycount. (2016, March 3). *Normal Map*. Retrieved June 26, 2017, from Polycount Wiki: http://wiki.polycount.com/wiki/Normal_map

Ramos, L. (2015). *Real-Time 3D ArchViz.* Porto: Instituto Superior de Tecnologias Avançadas.

Rawn, E. (2015, March 10). *Unreal Visualizations: 3 Pros and 3 Cons of Rendering with a Video Game Engine*. Retrieved January 30, 2017, from ArchDaily: http://www.archdaily.com/607849/unreal-visualizations-3-pros-and-3-cons-of-rendering-with-a-video-game-engine/

Rebelo, P. (2003, April 30). *3D Animation: An Introduction*. Retrieved May 5, 2017, from The University of Edinburgh: http://ddm.ace.ed.ac.uk/2002-03/pages/courses/augmentedreality/Lecture3/3D_Animation/

Reed, N. (2013, October 1). *Does HDR rendering have any benefits if bloom won't be applied?* Retrieved June 22, 2017, from Game Development StackExchange: https://gamedev.stackexchange.com/questions/62836/does-hdr-rendering-have-any-benefits-if-bloom-wont-be-applied

Robin. (2013, February 6). *Our Magic Tricks in 3D Asset Creation*. Retrieved April 30, 2017, from Klonk Games: http://www.klonk-games.com/2013/02/magic-asset-creation-tricks/

Russell, J. (2015, November 1). *Basic Theory of Physically Based Rendering*. Retrieved February 5, 2017, from Marmoset: https://www.marmoset.co/posts/basic-theory-of-physically-based-rendering/

Sanchez, M. (2016, September 6). *Progressing Beyond Pre-Render: The Marza Movie Pipeline for Unity*. Retrieved November 18, 2016, from Unity Blogs: https://blogs.unity3d.com/2016/09/06/progressing-beyond-pre-render-the-marza-movie-pipeline-for-unity/

Sanders, S. (2017). *Maya: Understanding Rigid & Soft Body Dynamics*. Retrieved June 26, 2017, from Steve's Digicams: http://www.steves-digicams.com/knowledge-center/how-tos/video-software/maya-understanding-rigid-soft-body-dynamics.html

sandy. (2014, December 15). *The 20 Best 3D Texturing & Painting Softwares*. Retrieved April 18, 2017, from RockThe3D: http://www.rockthe3d.com/20-best-3d-texturing-painting-softwares/

Shankar, D. (2017, February 22). *Lighting & Rendering in Unity 5*. Retrieved May 2, 2017, from Bigscreen Blog: https://blog.bigscreenvr.com/a-brief-overview-of-lighting-in-unity-5-fe21a68ee1d3

Shirley et al. (2013). *Basics of Physically-based Rendering.* Retrieved February 7, 2017, from Peter Shirley's papers and CV: http://www.cs.utah.edu/~shirley/papers/basics12.pdf

Side Effects Software Inc. (2017). *Unity Plug-in*. Retrieved April 26, 2017, from SideFX: https://www.sidefx.com/products/houdini-engine/unity-plug-in/

Slick, J. (2016, October 20). *7 Common Modeling Techniques for Film and Games*. Retrieved April 18, 2017, from Lifewire: https://www.lifewire.com/common-modeling-techniques-for-film-1953

Slick, J. (2016, November 2). *What is Rigging?* Retrieved May 2, 2017, from Lifewire: https://www.lifewire.com/what-is-rigging-2095

Slick, J. (2017, February 4). *What is Rendering?* Retrieved February 27, 2017, from Lifewire: https://www.lifewire.com/what-is-rendering-1954

Smit, R. (2012, August 24). *Top 3D Animation Software that Professionals Should Look At*. Retrieved April 18, 2017, from Pixelsmith Studios: http://pixelsmithstudios.com/blog/animation-articles/top-3d-animation-software-that-professionals-should-look-at

Tamnev, P. (2016, August 9). *Adam - Assets Creation for the Real Time Short Film*. Retrieved November 18, 2016, from CG Society: http://www.cgsociety.org/news/article/2587/adam---assets-creation-for-the-real-time-short-film

Tamnev, P. (2016, August 9). *ADAM - Assets Creation for the Real Time Short Film*. Retrieved April 20, 2017, from Unity Blogs: https://blogs.unity3d.com/2016/08/09/adam-assets-creation-for-the-real-time-short-film/

The Computer Language Company, Inc. (2016). *Definition of: GPU*. Retrieved June 26, 2017, from PC Magazine: https://www.pcmag.com/encyclopedia/term/43886/gpu

Theriault, M. (2014, September 29). *How to Make a Cutscene in Unity*. Retrieved May 6, 2017, from Cinema Suite: http://cinema-suite.com/how-to-make-a-cutscene-in-unity/

Tino. (2015, April 30). *Lightmapping in Unity 5*. Retrieved May 2, 2017, from Sassybot: https://sassybot.com/blog/lightmapping-in-unity-5/

Trammell, K. (2016, March 5). *Big Idea: "Baking"*. Retrieved June 26, 2017, from CG Cookie: https://cgcookie.com/2016/05/03/big-idea-baking/

Unite Europe. (2016, May). Making of The Gift: Game Changer in Film Production. Amsterdam, Netherlands. Retrieved June 21, 2017, from https://www.youtube.com/watch?v=d5XYNRH1nns

Unity Essentials. (2017, April 25). *Post Processing Stack*. Retrieved June 26, 2017, from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/83912

Unity Technologies. (2012, September 14). *Prefabs*. Retrieved April 29, 2017, from Unity Manual: https://docs.unity3d.com/400/Documentation/Manual/Prefabs.html

Unity Technologies. (2014). *Mecanim Animation System*. Retrieved June 26, 2017, from Unity Documentation: https://docs.unity3d.com/462/Documentation/Manual/MecanimAnimationSystem.htm l

Unity Technologies. (2016). *Adam*. Retrieved April 20, 2017, from Unity3D: https://unity3d.com/pages/adam

Unity Technologies. (2016). *Unity Manual*. Retrieved November 18, 2016, from Unity Documentation: https://docs.unity3d.com/Manual/index.html

Unity Technologies. (2017). *Ambient Lighting*. Retrieved May 2, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/ambient-lighting?playlist=17102

Unity Technologies. (2017, April 21). *Art Asset Best Practice Guide*. Retrieved May 5, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/HOWTO-ArtAssetBestPracticeGuide.html

Unity Technologies. (2017). *Choosing a Color Space*. Retrieved May 1, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/choosing-color-space?playlist=17102

Unity Technologies. (2017). *Choosing a Lighting Technique*. Retrieved May 1, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/choosing-lighting-technique?playlist=17102

Unity Technologies. (2017). *Choosing a Rendering Path*. Retrieved May 1, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/choosing-rendering-path?playlist=17102

Unity Technologies. (2017, March 29). *Color Correction Lookup Texture*. Retrieved May 6, 2017, from Unity Documentation: https://docs.unity3d.com/550/Documentation/Manual/script-ColorCorrectionLookup.html

Unity Technologies. (2017). *Emissive Materials*. Retrieved May 2, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/emissive-materials?playlist=17102

Unity Technologies. (2017). *High Dynamic Range (HDR)*. Retrieved May 1, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/high-dynamic-range-hdr?playlist=17102

Unity Technologies. (2017). *Introduction to Lighting and Rendering*. Retrieved May 1, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/introduction-lighting-and-rendering?playlist=17102

Unity Technologies. (2017). *Light Probes*. Retrieved May 2, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/light-probes?playlist=17102

Unity Technologies. (2017). *Light Types*. Retrieved May 2, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/light-types?playlist=17102

Unity Technologies. (2017, April 21). *Prefabs*. Retrieved April 29, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/Prefabs.html

Unity Technologies. (2017, April 21). *Procedural Mesh Geometry*. Retrieved April 26, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/GeneratingMeshGeometryProcedurally.html

Unity Technologies. (2017, April 21). *ProceduralTexture*. Retrieved April 26, 2017, from Unity Documentation: https://docs.unity3d.com/ScriptReference/ProceduralTexture.html

Unity Technologies. (2017). *Reflections*. Retrieved May 1, 2017, from Unity Tutorials: https://unity3d.com/learn/tutorials/topics/graphics/reflections?playlist=17102

Unity Technologies. (2017, April 21). *Static GameObjects*. Retrieved April 29, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/StaticObjects.html

Unity Technologies. (2017, May 5). *The Animator Controller Asset*. Retrieved May 6, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/Animator.html

Unity Technologies. (2017, April 21). *Unity's Rendering Pipeline*. Retrieved May 2, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/SL-RenderPipeline.html

Unity Technologies. (2017, April 21). *Using the Mesh Class*. Retrieved April 26, 2017, from Unity Documentation: https://docs.unity3d.com/Manual/UsingtheMeshClass.html

Vardanega, J. (2013). *Pixar's Animation Process*. Retrieved February 27, 2017, from Pixar: http://pixar-animation.weebly.com/pixars-animation-process.html

Vetasoft. (2017, June 15). *Camera Filter Pack*. Retrieved June 26, 2017, from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/18433

WebWise Team. (2012, October 10). *What are plug-ins?* Retrieved June 26, 2017, from BBC WebWise: http://www.bbc.co.uk/webwise/guides/about-plugins

Wilson, J. (2015, October 1). *Physically-Based Rendering, And You Can Too!* Retrieved February 5, 2017, from Marmoset: https://www.marmoset.co/posts/physically-based-rendering-and-you-can-too/

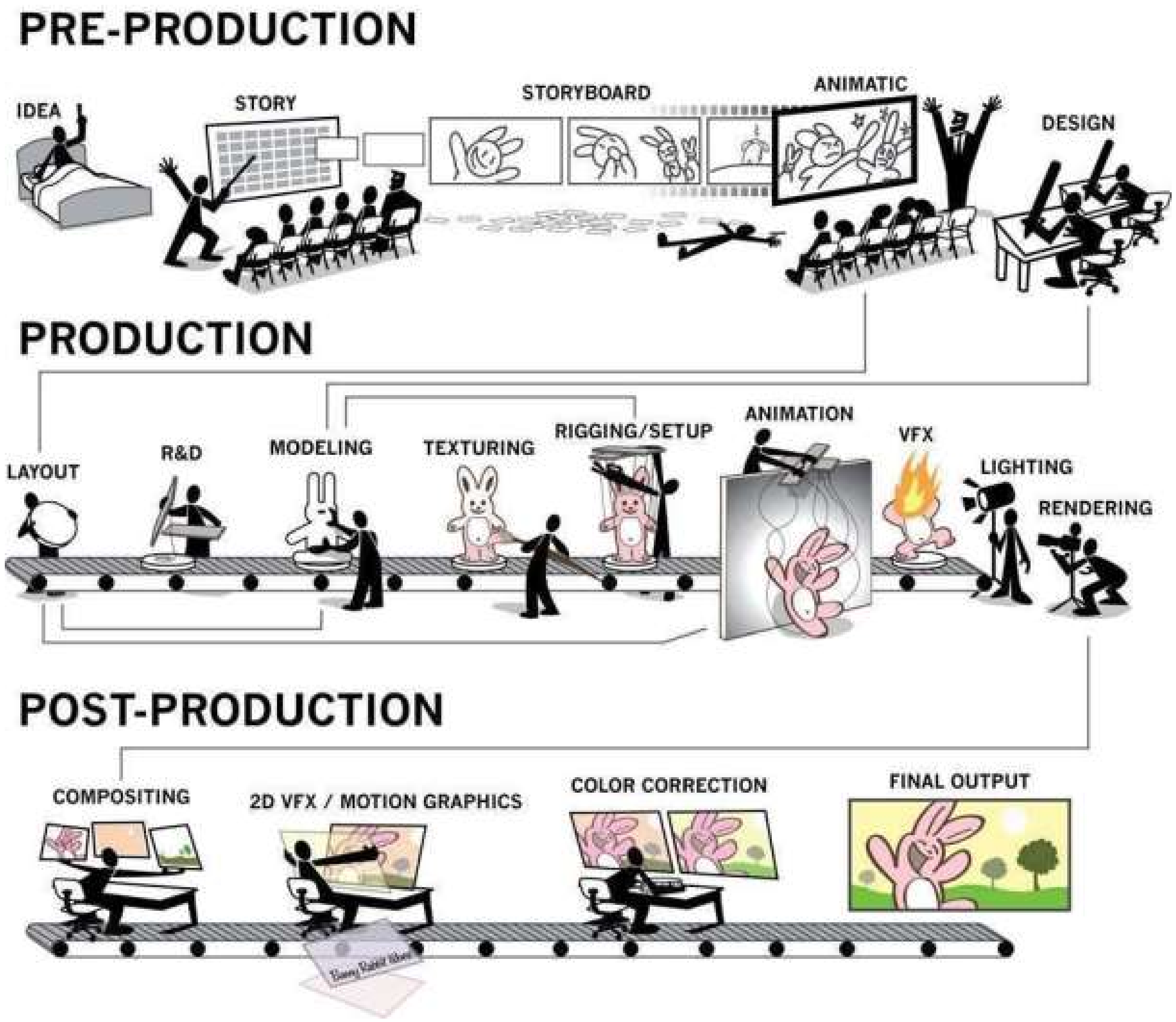
Wolfe, J. (2012, February 17). *Autodesk Tools Used on 14 Oscar-Nominated Films*. Retrieved February 17, 2017, from Animation World Network: http://www.awn.com/news/autodesk-tools-used-14-oscar-nominated-films

You can do it! VFX. (2017, April 3). *Offline Render*. Retrieved May 2, 2017, from Unity Asset Store: https://www.assetstore.unity3d.com/en/#!/content/60925

Zucconi, A. (2015, June 10). *A gentle introduction to shaders in Unity3D*. Retrieved June 26, 2017, from Alan Zucconi: http://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/

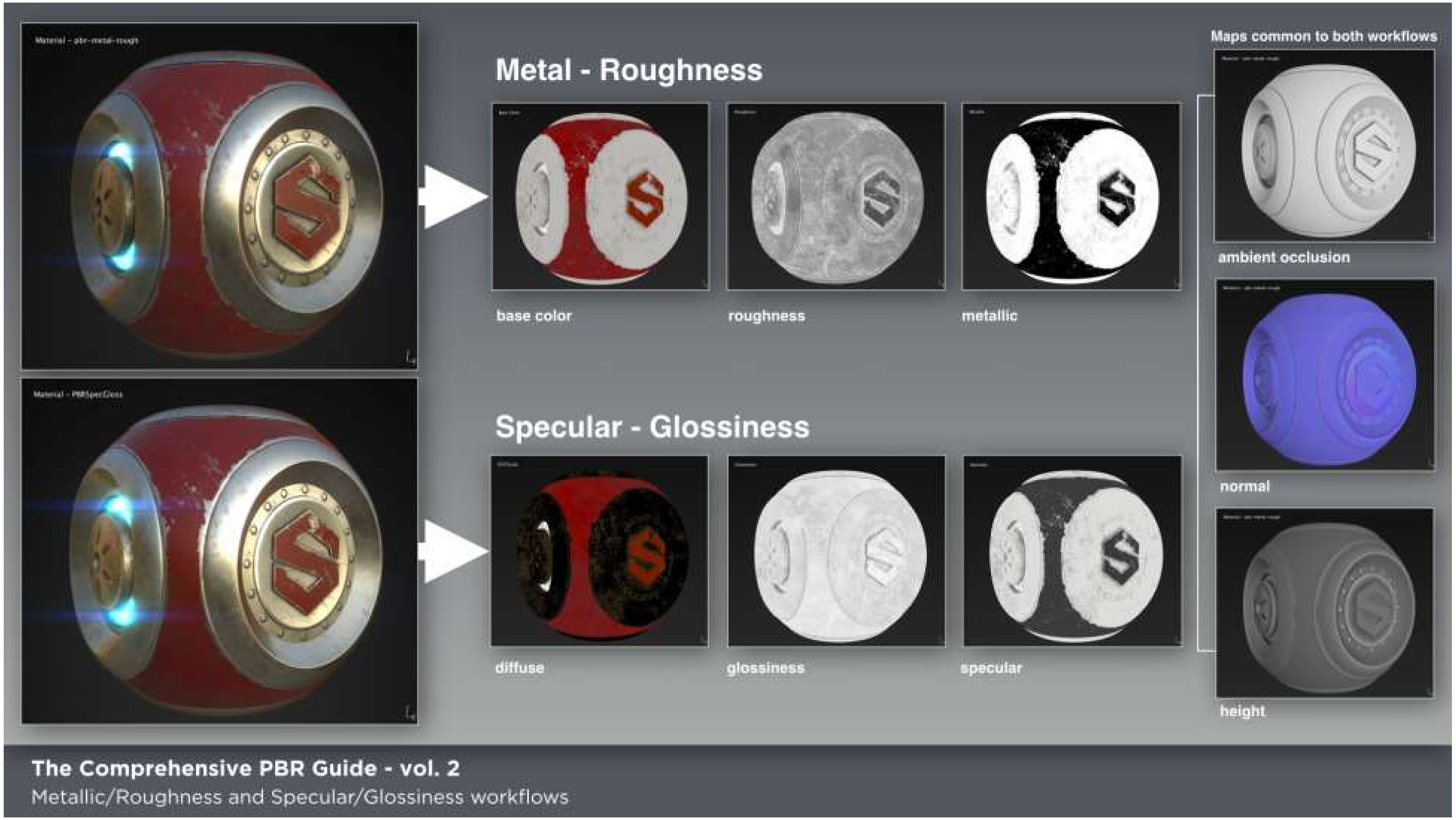
Zucconi, A. (2015, July 8). *Screen shaders and image effects in Unity3D*. Retrieved June 26, 2017, from Alan Zucconi: http://www.alanzucconi.com/2015/07/08/screen-shaders-and-postprocessing-effects-in-unity3d/

ANNEX A – 3D Production Pipeline



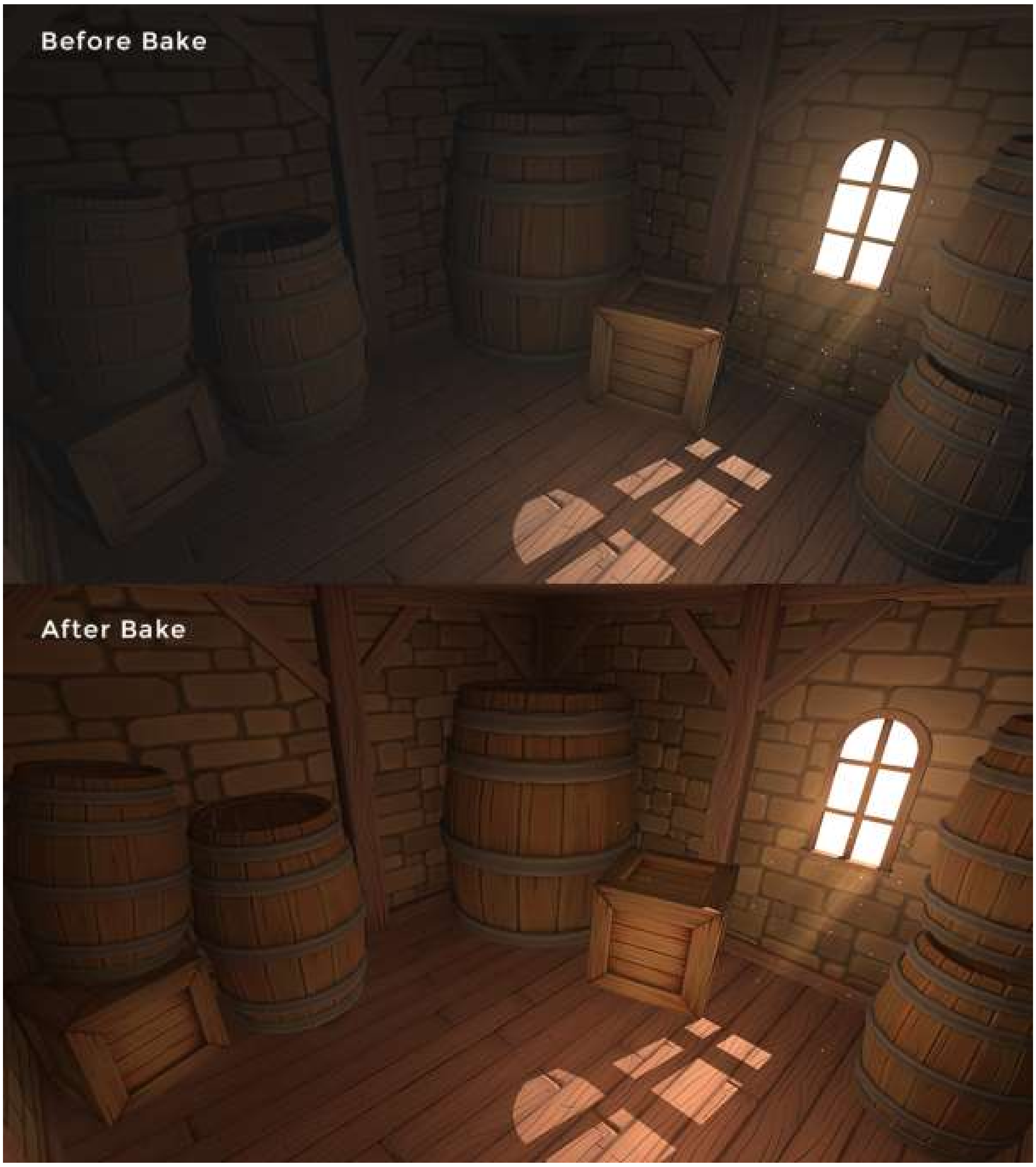
Annex A - 3D Production Pipeline (Ludwick apud *Beane, 2012, p. 23)*

ANNEX B – PBR Workflows



Annex B - PBR Workflows (McDermott, The Comprehensive PBR Guide by Allegorithmic - vol. 2, Light and Matter: Practical Guidelines for Creating PBR Textures, 2015, p. 2)

ANNEX C – Before and After Light Baking in Unity



Annex C – Before and after Light Baking in Unity (Grove, 2016, para. 6)