RENDERER CONVERSION TOOL FOR AUTODESK MAYA

Master's Project Thesis

Shubham Prabhakar

s5512613

M.Sc. Computer Animation and Visual Effects

Date: 14th August, 2023

Abstract

The fast-developing field of 3D rendering necessitates practical tools to switch between several render engines smoothly, allowing artists to explore new creative directions without laboriously redoing scenes. This thesis presents a pioneering solution—a Renderer Conversion Tool for Autodesk Maya designed to bridge the gap between diverse rendering platforms. The programme gives artists a helpful head start by instantly converting scenes from one renderer to another, utilising precisely curated vocabulary files that control the conversion process. While not achieving perfection, this tool speeds up the process and allows artists to fine-tune attributes after conversion, improving their workflow quickly.

Regardless of their technical expertise, all 3D artists can utilise the tool thanks to its user-friendly design. The programme's inner workings are revealed by thoroughly investigating its technique, revealing the crucial part dictionary files play in coordinating the conversion. The knowledge needed to coordinate materials, textures, and lights across several render engines is contained in these files. The tool supports Autodesk's Arnold renderer, Pixar's Renderman, and Chaos Group's V-Ray renderer. However, thanks to its adaptable architecture, it may easily incorporate more renderers through dictionary file extensions in the future.

Acknowledgement

I would like to thank my supervisor, Professor Jon Macey, for his suggestions and direction in helping me to complete my project, as well as the rest of the National Centre for Computer Animation's teaching staff, including Jian Chang, Jian Jun Zhang, Phil Spicer, Ian Stephenson, Lihua You, and Veno Prendergast. They have all assisted me in developing my skills throughout the year.

I also want to thank Ishita Sehgal, Yashasvi Yederi, Vaibhav and Zoe Thomas, all fellow NCCA students, for utilising my tool and providing me with insightful feedback.

Finally, I want to thank my friends and family for always being there for me.

Table of Contents

1 Introduction

The choice of a rendering engine significantly impacts the final visual result in the constantly changing world of 3D graphics and animation. Each rendering engine's distinctive features, strengths, and subtleties influence the unique look and feel of produced scenes. However, for 3D artists and content producers, the intrinsic diversity of rendering systems poses a significant obstacle. Maintaining the integrity of the original scene when switching between render engines flawlessly has become increasingly urgent. This thesis offers an innovative Autodesk Maya conversion tool that successfully addresses the issue. This adaptable approach makes converting scenes between supported render engines rapid and straightforward. The combination of cutting-edge technology, a simple user interface, and extensive dictionary files that codify the subtleties of rendering engine properties forms the basis of the application. This technology is an essential part of the toolbox of today's 3D content creators since it enables quick and accurate conversions while empowering artists to make fine tweaks after conversion.

2 Motivation

The motivation behind this project arose when I was working with my fellow NCCA students, and they needed to learn how to use Pixar's Renderman to render our scene as we were going to use it to achieve more photorealism. They only knew how to use Arnold renderer in Autodesk Maya.

The field of 3D graphics has been dramatically transformed by advanced rendering engines like Autodesk Arnold, Pixar's Renderman and Chaos Group's V-Ray, to name a few, which provide many tools to help achieve various visual aesthetics. However, switching a scene from one engine to another frequently requires time-consuming and extensive conversion effort due to the disparity in vocabulary, parameterization, and algorithms among different engines. Such conversions have typically required significant effort, requiring artists to carefully rebuild shaders, textures, and lighting setups, even though they are crucial for workflow flexibility and artistic expression.

The main goal of this project is to create, develop, and put into practice a tool that makes switching between several rendering engines within the Autodesk Maya software environment easier. This utility depends on dictionary files containing the data required for attribute mapping across various rendering engines. The goal of this research is to balance user control

and automation. At the same time, allowing artists to fine-tune the outcomes to reflect their artistic vision, the conversion process is meant to be accelerated.

3 Structure of the Thesis

The development, use, and ramifications of the Renderer Conversion Tool for Autodesk Maya are thoroughly explored in this thesis. I will discuss my previous work, tools available online and the limitations of both my work and these tools in Chapter 4. In Chapter 5, the fundamental software and rendering engines, that are, Autodesk Maya, Autodesk Arnold, Pixar's Renderman, and Chaos Group's V-Ray - are thoroughly discussed, with unique features and difficulties highlighted. The development of the tool, its architecture, user interface, and use of dictionary files are all explained in Chapter 6. In Chapter 7, the tool's capabilities are explored in detail and demonstrating how it is used. Reviews are presented in Chapter 8 to show how the tool can be used in realistic situations and feedback from artists. Finally, the tool's broader ramifications, prospective upgrades, directions for further research and conclusion are covered in Chapter 9.

4 Previous Work and Research

In this section, I will discuss the prior efforts I undertook to address the challenge of making this tool, the evaluation of existing tools available on the internet and the limitations of both my work and these tools.

4.1 Pipeline and TD assignment - Automatic Material Swapper

The fundamental differences in shader systems, parameterisation, and rendering algorithms make converting materials between rendering engines, such as from Arnold to Renderman, difficult. In the early stage of my research and our group project, I came across the requirement to permit smooth transitions between these two well-known renderers, motivated by the desire to access Renderman's photorealistic advantages. I made a tool for our Pipeline and Technical Direction unit that automatically swaps between Autodesk's Arnold and Pixar's Renderman materials and vice versa in Maya (Prabhakar 2023).

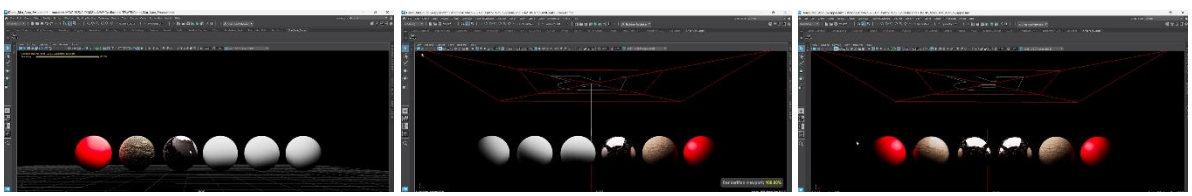

Figure 1, 2 and 3: Automatic Material Swapper (Prabhakar 2023)

The limitations of this tool were that it only supported one material each, aiStandardSurface for Arnold and PxrSurface for Renderman. It also only matched some attributes between the renderers. The main issue with this tool was that it was an ad hoc solution and was difficult to extend, as per the feedback received from my professor Jon Macey.

4.2 Evaluation of existing tools

As part of my research, I surveyed existing tools available on the internet designed to tackle renderer conversions between different renderers. I found a tool - Convert Renderer by create3dcharacters.com that will instantly convert a renderer between our three supported renderers, Arnold, Redshift and Renderman. Limitations apply. (Create3dcharacters 2021) The tool converts shaders and lights to the native renderer node types. (Silke 2021) However, this tool, too, had limitations. It did not support displacement, normal map, bump map and metalness attributes, and it only converted three types of lights. (Create3dcharacters 2021)

I found another tool - Maya Scene Converter by Mahmoud El-Ashry. (mhdmhd 2021) Maya Scene Converter is a Python-based Qt tool for converting scenes from one render engine to another in Maya. (lesterbanks 2020) While it is not a perfect one-shot conversion, the Maya tool relies on rules to automate the process. However, this tool, too, had limitations. This tool required different rule files for every renderer conversion, which meant adding a new one for every new renderer support. It also did not support Pixar's Renderman.

4.3 Lessons Learned and Building Blocks

I could comprehend the complex nature of renderer conversions thanks to the expertise I obtained from building and working on my Pipeline and TD assignment project and researching the existing tools. It emphasised the importance of comprehensive solutions that offer user-friendly interfaces and cover a broader range of renderers. The knowledge gathered during this study phase served as the basis for constructing my Renderer Conversion Tool for Autodesk Maya, which is the subject of this thesis.

5 Technical Background

This chapter delves deeply into the complex technological foundations of the essential software and rendering engines at the heart of our investigation. Autodesk Maya, Autodesk Arnold, Pixar's Renderman, and Chaos Group's V-Ray are thoroughly examined to understand their unique traits, potential, and difficulties.

5.1 Autodesk Maya

A key player in 3D graphics and animation, Autodesk Maya provides a comprehensive set of tools necessary for content development. Maya is professional 3D software for creating realistic characters and blockbuster-worthy effects. (Autodesk 2021a) Users can build complex scenarios and generate lifelike animations, realistic textures, and final renders because of the software's versatility.

Maya's node-based architecture offers a modular and non-destructive workflow in which a network of nodes connects diverse components. This method promotes creativity by enabling artists to experiment with various settings and parameters and also helps the programmers for writing scripts.

As mentioned in the Autodesk Maya 2024 documentation, Maya is built around nodes. An "object", such as a sphere, is built from several nodes (Autodesk 2023):

- a creation node that records the options that created the sphere (Autodesk 2023)
- a transform node that records how the object is moved, rotated, and scaled (Autodesk 2023)
- and a shape node that stores the positions of the spheres control points (Autodesk 2023)

For example, if you select Create > NURBS Primitives > Sphere to create a sphere, Maya creates a transform node and a shape node(Autodesk 2023).

The sphere's shape node holds the mathematical description of the sphere's shape(Autodesk 2023). The sphere's transform node holds the sphere's position, scaling, rotation, and so on. The shape node is the child of the transform node. (Autodesk 2023)

If you select Options > Display > Shape Nodes in the Hypergraph, the scene hierarchy shows these nodes for the sphere: (Autodesk 2023)

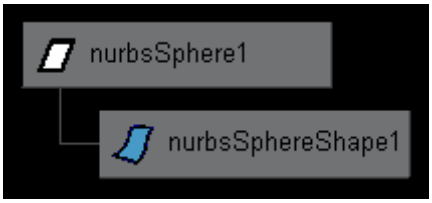

Figure 4: Scene hierarchy (Autodesk 2023)

Attributes are also crucial for scripting in Autodesk Maya. The numerous settings and factors that determine the characteristics of an item, component, or element inside a scene are

attributes. These characteristics determine how an object appears, functions, and engages with surrounding elements, like lights, cameras, and materials.

As mentioned in Autodesk Maya 2024 documentation, an attribute is a position associated with a node that can hold a value or a connection to another node. (Autodesk 2023) Attributes control how a node works(Autodesk 2023). For example, a transform node has attributes for the amount of rotation in X, Y, and Z. (Autodesk 2023) You can set attributes to control practically every aspect of your animation. (Autodesk 2023)

There are many ways to set attributes in Maya: with the Attribute Editor, the Channel Box, the Attribute Spread Sheet, menu selections, and MEL. (Autodesk 2023)

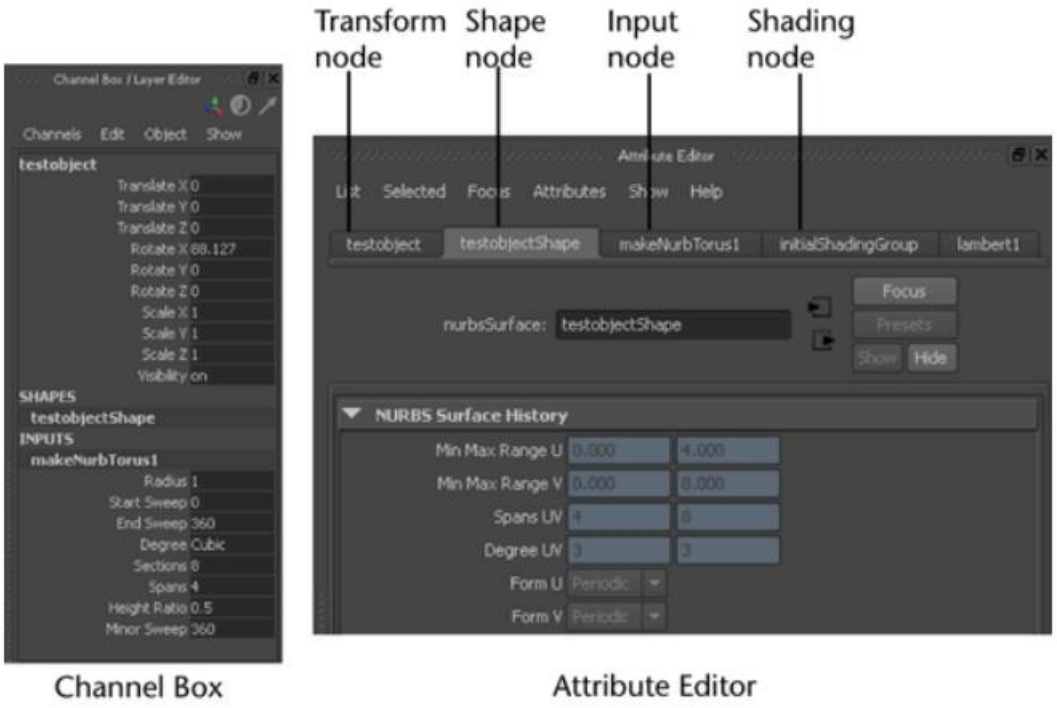

Figure 5: Maya Channel Box and Attribute Editor (Autodesk 2023)

However, Maya's thoroughness may need to be improved. Complex scene management and obtaining the fastest render times require effective resource management. Complexity and usability must remain balanced, especially for new users navigating the complicated toolkit.

Now, we will be examining all the different renderers I have worked on, especially the lights and materials of these renderers.

5.2 Autodesk Arnold

Autodesk Arnold stands out as a pioneer in the field of physically based rendering engines, praised for its accuracy in modelling light interactions and producing photorealistic results. Arnold uses path-tracing techniques to carefully capture global illumination, reflections, refractions, and other subtle phenomena.

Some of the most essential features of Arnold are:

- Arnold GPU - Switch seamlessly between CPU and GPU rendering. (Autodesk 2021b)
- Adaptive sampling - Use another means of tuning images to reduce render times without jeopardizing final image quality. (Autodesk 2021b)
- Subsurface scatter - High-performance ray-traced subsurface scattering eliminates the need to tune point clouds. (Autodesk 2021b)
- Open Shading Language (OSL) support - Use Open Shading Language (OSL), an advanced shading language for Global Illumination renderers. (Autodesk 2021b)
- Denoising - Powerful denoising solutions offer you the flexibility to use much lower-quality sampling settings. (Autodesk 2021b)

The most defining feature is Arnold's user-friendly interface, which streamlines the rendering process. Therefore, many beginner artists prefer Arnold for rendering their work. However, its underlying complexity caters to experienced users who may adjust settings to obtain the best results. However, to fully realise Arnold's potential, one must be intimately versed in its complexities, such as shader networks and complex lighting setups.

Arnold Lights

Each form of light that Arnold delivers has unique characteristics that regulate how the light affects the surroundings and objects. The most typical lighting fixtures in Arnold are:

- Area light (including support for disk, cylinder, quad)
- Sky Dome light
- Mesh light
- Directional light
- Point light
- Spot light

Some of the essential attributes of all these lights which enable the artist to customise their scene’s illumination are given below:

- Intensity – It determines the strength of the light emitted by the light source
- Color – It determines the hue of the light emitted by the light source.
- Exposure – It is a f-stop value that adjusts the overall brightness of the Arnold light source.
- Visibility – It determines whether the light is visible to the camera.
- Shadows – It controls whether the light casts shadows in the scene.

Note:

In Arnold as per official Autodesk Arnold documentation, the total intensity of the light is computed with the following formula: (Autodesk 2023)

color * intensity * 2^exposure (Autodesk 2023)

You can get the same output by modifying either the intensity or the exposure. For example, intensity=1, exposure=4 is the same as intensity=16, exposure=0. Note: 2 0 = 1, not 0. (Autodesk 2023)

1 * 1 * 2^4 = 16

1 * 16 * 2^0 = 16

Additionally, Arnold's lights smoothly combine with other rendering components, interacting with materials and textures realistically, adding to global lighting and creating realistic reflections and refractions. Artists may effectively control the mood, ambience, and visual narrative of their scenes by adjusting these factors, producing realistic and visually arresting results.

Arnold Materials

Arnold materials are essential for determining how things appear visually in 3D by modelling how light interacts with their surfaces. The adaptability and realism of Arnold's material system are well known.

The most commonly used materials in Arnold are:

- aiStandardSurface

  The "aiStandardSurface" is one of Arnold's most prominent material shaders. This physically-based shader mimics various real-world materials, including metals, plastics,

textiles, and skin. (Arnold 2023) Artists can alter a wide range of its features to produce precise material properties.

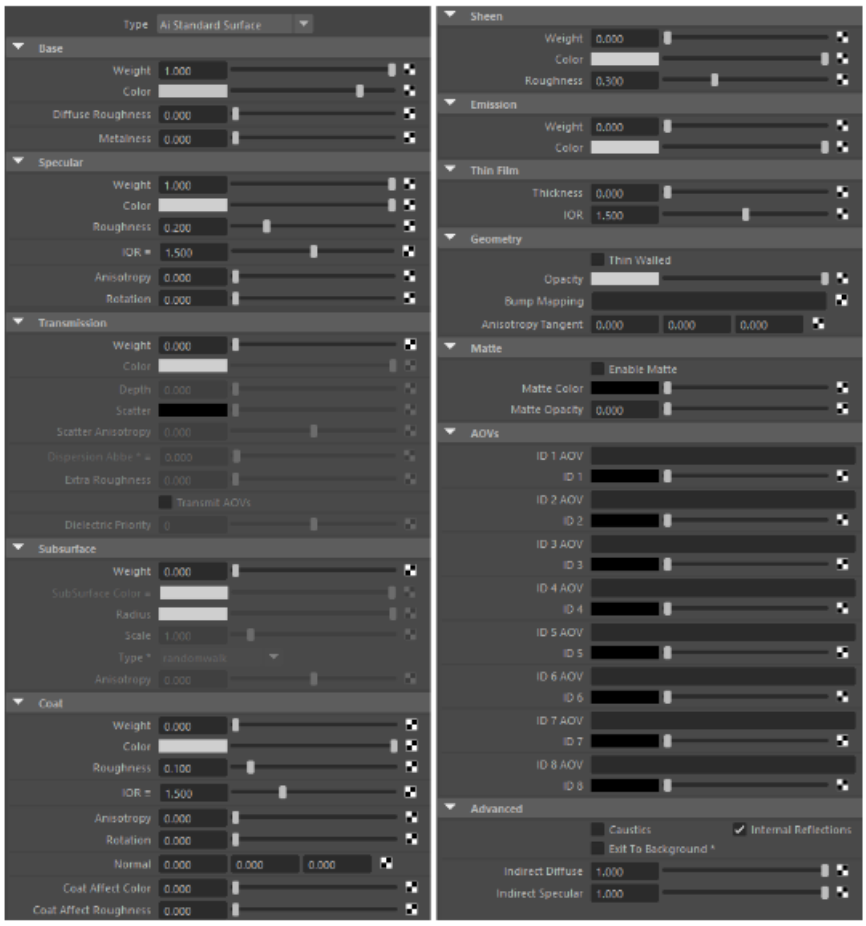

Figure 6: aiStandardSurface attributes (Arnold 2023)

It also comes with many standard surface presents like blood, car paint, ceramic, plastic, rubber, wax and many more. (Arnold 2023)

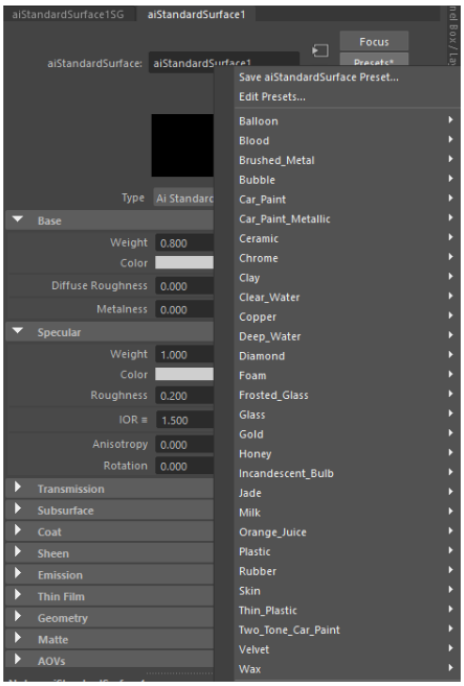

Figure 7: aiStandardSurface presets (Arnold 2023)

- aiStandardHair

The "aiStandardHair" shader in Arnold also provides extensive features for rendering hair and fur. Due to the specific ways that hair and fur interact with light, they pose particular difficulties. This shader is designed to simulate the intricate relationships between light and hair strands, resulting in an accurate and eye-catching representation of hair and fur.

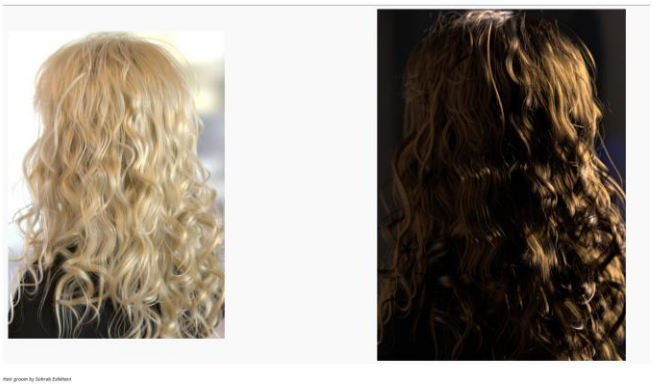

Figure 8: aiStandardHair render (Arnold 2023)

Some important attributes of Arnold materials are listed below:

- Base color - This attribute defines the overall color of the material.

- Metalness – This attribute controls the balance between a surface's dielectric (non-metal) and metallic behavior.
- Specularity – It determines the intensity of the material's specular reflections.
- Transmission - This attribute handles light transmission through transparent or translucent surfaces.
- Subsurface - It simulates how light penetrates and scatters within translucent materials, like skin
- Coat – It simulates a secondary layer of material on top of the primary surface.
- Sheen - It mimics the velvet-like appearance of certain surfaces
- Emission – It controls the material's self-illumination, allowing it to emit light.
- Normal Map – This helps in adding surface detail without altering geometry.
- Specular Tint – It is the scale of the primary specular contribution, which multiplies the primary specular color. (Arnold 2023)
- 2nd Specular Tint – It is the scale of the secondary specular contribution, which simply multiplies the secondary specular color. (Arnold 2023)
- Transmission Tint – It is the scale of the transmission contribution, which simply multiplies the transmission tint. (Arnold 2023)

The "aiStandardHair" and "aiStandardSurface" shaders perfectly exemplify Arnold's dedication to realism and aesthetic freedom. With the help of their extensive attribute sets, artists can produce materials that closely mimic their real-world equivalents, allowing them to create genuine and visually arresting situations that attract viewers.

5.3 Pixar Renderman

Pixar Animation Studios created Pixar's RenderMan, a groundbreaking and top-tier rendering software. RenderMan is renowned for its extraordinary ability to create cinematic graphics of the highest caliber and has been instrumental in advancing computer graphics and animation. It provides a wide range of tools and capabilities that enable artists and studios to produce realistic artwork that is visually appealing. Advanced algorithms that replicate light behavior are the foundation of the rendering engine used by RenderMan, which enables precise light interactions, shadow casting, reflections, and refractions.

RenderMan is an essential component of the animation pipeline because of its versatility, which goes beyond rendering and includes various shading, lighting, and special effects tools. Thanks to its long history and ongoing dedication to innovation, RenderMan remains a top choice for

studios and artists looking for uncompromised quality and creative freedom in their visual projects.

Pixar Lights

Lighting in RenderMan is flexible but also very straightforward for users. RenderMan includes a set of basic geometric lights, built-in. (Pixar 2023) These specialized lights are called the following:

PxrDomeLight

PxrRectLight

PxrDistantLight

PxrDiskLight

PxrSphereLight

PxrMeshLight

The crucial parameters of all the different types of lights in Renderman are:

- Intensity – Scales the contribution of this light linearly. (Pixar 2023)
- ColorMap - An image to use as a light source. Preferably a HDRI. (Pixar 2023)
- Color – The color of the emitted light. (Pixar 2023)
- Exposure – Specifies the exposure of the area light as a power of 2. (Pixar 2023)
- Primary Visibility – light isn't directly visible to the camera by default. Turning this on makes it visible. (Pixar 2023)
- Enable Shadows – Enable raytraced shadows. (Pixar 2023)

Specific non-physical controls are designed to assist artists in creating art-directed imagery by disregarding specific laws of physics that are typically simulated. The PxrSphereLight requires that all scale transforms be uniform, e. g. X,Y, and Z scale must all be the same so as not to skew the sphere shape into an ellipse or other form. (Pixar 2023) Lights are not visible in refraction/transmission by default. This can be enabled in the light visibility settings. (Pixar 2023) The exposure of the lights works the same way it works in the case of Arnold lights.

Pixar Materials

Materials and their characteristics are crucial in RenderMan from Pixar to produce the realistic and eye-catching effects for which the programme is known. Two of the most used materials are:

- PxrDisneyBSDF

This Bxdf is based on Brent Burley's extended material model, introduced in "Extending the Disney BRDF to a BSDF with Integrated Subsurface Scattering" at SIGGRAPH 2015. (Pixar 2023)

The PxrDisneyBsdf is a physically-based model that is simple and effective, with only a few parameters needed to create various photorealistic and artistic materials.

Figure 9: Teapot with PxrDisneyBSDF (Pixar 2023)

- PxrMarschnerHair

PxrMarschner is a material designed to render realistic hair, fur, and fibres. (Pixar 2023) With properties that govern not only colour and transparency but also the complex way light behaves when it interacts with individual hair strands, this shader is explicitly designed for the complexity of hair.

© Disney/Pixar

Figure 10: Characters with PxrMarschnerHair (Pixar 2023)

Some attributes of Pixar materials are listed below:

- Base color - This attribute defines the overall color of the material.
- Metallic – This attribute controls the balance between a surface's dielectric (non-metal) and metallic behavior.
- Specularity – It determines the intensity of the material's specular reflections.
- Subsurface - It simulates how light penetrates and scatters within translucent materials, like skin
- Coat – It simulates a secondary layer of material on top of the primary surface.
- Sheen - It mimics the velvet-like appearance of certain surfaces
- Emission – It controls the material's self-illumination, allowing it to emit light.
- Normal Map – This helps in adding surface detail without altering geometry.
- SpecularGainR – Gain for the R lobe of Marschner specular. This is like a clearcoat where the specular is fairly sharp and glossy and normally not colored. (Pixar 2023)
- SpecularGainTRT – Gain for the TRT lobe of Marschner specular. This is a rougher and colored specular. This goes through the hair fiber volume twice. (Pixar 2023)
- SpecularGainTT - Gain for the TT lobe of Marschner specular. This is a transmission-type (refraction) specular with some volume attenuation. (Pixar 2023)

RenderMan is recognised as a leader in the industry for its dedication to accurately rendering the surface characteristics and the light interplay with these complex materials.

This combination of qualities gives artists a dynamic arsenal that allows them to do more than replicate the tangible but also to breathe life into textures and materials.

5.4 Chaos Group's V-Ray

V-Ray, a renowned and well-known rendering engine developed by Chaos Group, has substantially influenced computer graphics. V-Ray, renowned for its extraordinary ability to combine realism with versatility, has established itself as a critical component in producing top-notch visual effects. V-Ray excels at capturing the subtle interplay between illumination and materials because it emphasises accurate light simulation, producing images that closely resemble real-world scenes. V-Ray's adaptability is shown by its easy connection with the most popular 3D software programmes, giving artists a straightforward and effective workflow.

V-Ray Lights

V-Ray lights are created especially for V-Ray to maximise its functionality. While Maya regular lights can be used with V-Ray, the software also has a set of lights explicitly made for rendering with the V-Ray engine. These explicit lights are:

- VRayLightRectShape – equivalent to area light in Arnold
- VRayLightSphereShape – equivalent to point light in Arnold
- VRaySunShape – equivalent to directional light in Arnold
- VRayIESShape – equivalent to spot light in Arnold
- VRayLightDomeShape – equivalent to sky dome light in Arnold
- VRayLightMesh – equivalent to mesh light in Arnold

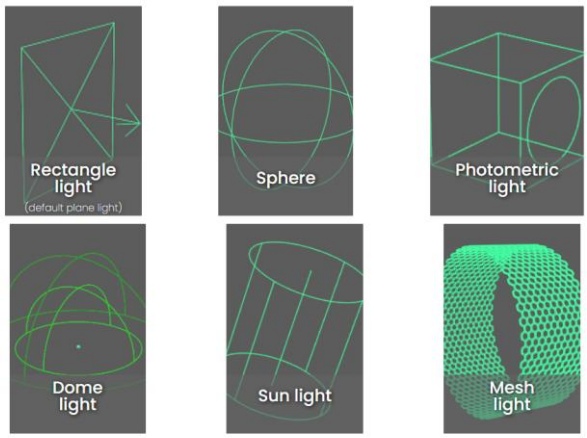

Figure 11: Different types of lights in V-Ray (Chaos 2023)

These lights' characteristics, such as intensity, colour, visibility, and shadows, are comparable in Autodesk Arnold, Pixar's RenderMan, and Chaos Group's V-Ray. The shared attribute consistency highlights the industry's desire for a consistent approach to lighting across different renderers. It is crucial to remember that although these qualities are very similar, there are some differences. Notably, V-Ray lights differ from their counterparts in lacking the 'Exposure' feature. As a result, when converting lighting configurations, a modification is required. In order to provide a consistent lighting arrangement after the successful conversion of the lights, this adaption entails converting the values of the intensity characteristic. This discrepancy emphasises the complexity of synchronising lighting parameters between various rendering engines, necessitating a thorough comprehension of each engine's characteristics and behaviour to ensure correct and reliable results throughout the conversion process.

V-Ray Materials

The robust material system in Chaos Group's V-Ray is essential for attaining the highest level of realism and visual impact in rendered scenes. These materials are quite versatile, and each can be used in several ways to achieve different looks as needed, from simulating simple surface properties like plastics and metals to simulating complex uses such as translucent objects and even light-emitting objects. (Chaos 2023) The most commonly used materials are:

- VRayMtl

The "VRayMtl" is a cornerstone of its material offerings. Using this shader's rich properties, artists can create a wide range of materials, from metals and plastics to textiles and glass.

Furthermore, with the VRayMtl you can apply different texture maps, control the reflections and refractions, add bump and displacement maps, force direct GI calculations, and choose the BRDF for how light interacts with the surface material. (Chaos 2023)

Image courtesy of Mathew Monro

Figure 12: V-Ray Render with VrayMtl (Chaos 2023)

- VRayHairNextMtl

The "VRayHairNextMtl" is designed exclusively for rendering hair and fur. Its characteristics are painstakingly designed to capture the fine details of hair. It offers options for a workflow based on the physiological characteristics of actual hair. This material employs a straightforward melanin slider to control the hair colour like in the real world, rather than adjusting arbitrary colours that blend.

Figure 13: V-Ray Render with VrayHairNextMtl (Chaos 2023)

Some attributes of V-Ray materials common with Arnold and Renderman are:

- Base color – Similar to base color in Arnold and Renderman.
- Metalness – Similar to metalness in Arnold and metallic in Renderman.
- Reflection – Similar to specularity in Arnold and Renderman.
- Coat – Similar to coat in Arnold and Renderman.
- Sheen - Similar to sheen in Arnold and Renderman.
- Illumination – Similar to emission in Arnold and Renderman.
- Bump Map – Similar to normalCamera in Arnold and bumpNormal in Renderman.
- Primary Tint – Similar to specularTint and specularGainR in Arnold and Renderman
- Secondary Tint – Similar to specular2Tint and specularGainTRT in Arnold and Renderman
- Transmission Tint – Similar to transmissionTint and specularGainTT in Arnold and Renderman

The comparison of different rendering engines reveals various strengths, each suited to various aesthetic goals. Precision and realism are strengths of Arnold, creative control of Renderman, and adaptability of V-Ray. However, due to variations in terminology and algorithms, there still needs to be a hurdle in achieving seamless translation between these engines.

6 Development

6.1 Tool Architecture

I thoroughly investigated Autodesk Maya and the various rendering engines under consideration, guided by a thorough review of the literature and my prior work. The investigation's conclusion was a clear implementation plan for the Renderer Conversion Tool. I decided on a modular strategy because I needed a flexible and adaptable framework, and it created the groundwork for future expansion and potential improvements.

Modular Approach for Scalability

The decision to use a modular architecture is based on its capacity to provide an integrated yet adaptable structure. My code contains unique functions, promoting code modularity and making future modifications easier. This strategy allows additional rendering engines to be easily included in the tool's architecture as they are developed. The tool architecture facilitates simple maintenance, extensibility, and debuggability by compartmentalizing components like attribute mapping, conversion logic, and user interface interactions.

User-Friendly Interface

I concentrated on building an easy-to-use user interface (UI) since I knew its importance in improving user experience. No matter how technically skilled an artist is, the UI is designed to be usable and accessible. This design philosophy considers the artists' various skill levels in a pipeline, ensuring that both beginners and specialists may use the tool to its full potential. The user interface (UI), which offers choices for choosing source and target engines, customizing conversion settings, and previewing converted scenes, is a portal to the tool's robust conversion capabilities.

6.2 Dictionary Files

I started making dictionary files, a crucial part of the Renderer Conversion Tool, after thoroughly understanding the distinctive properties, materials, and lighting available in various rendering engines. These files, with the filename extensions "lights.json" and "materials.json," constitute the basis for seamless attribute mapping across various rendering systems.

Attribute Mapping and Explanation

The dictionary files are painstakingly designed to provide a simple and direct translation between the properties of various renderers. Each file entry outlines how specific material or light-type attributes in one rendering engine correspond to their equivalents in another. The attribute mapping is supported by thorough justifications that guarantee a comprehensive comprehension of the functions and effects of the attributes.

Flexibility and Scalability

These dictionary files' inherent adaptability and scalability are defining characteristics. They are made to accept the integration of new rendering engines easily. Its ability to integrate new render engines with their attribute definitions as the rendering landscape changes maintain the tool's versatility. This architectural foresight guarantees that the tool will be current and valuable while accommodating new rendering technology.

These files are given below:

lights.json

```
{
    "Lights" :
    {
        "renderer" : ["arnold","renderman","vray"],
```

```
"area_light" : {
    "renderer" : ["arnold","renderman","vray"],
    "name" : ["aiAreaLight","PxrRectLight","VRayLightRectShape"],
    "attribute_intensity" : ["intensity","intensity","intensity"],
    "attribute_color" : ["color","lightColor","color"],
    "attribute_exposure" : ["exposure","exposure",""],
    "attribute_visibility" : ["aiCamera","visibility","visibility"],
    "attribute_shadows" :["aiCastShadows","enableShadows","shadows"]
},
"point_light" : {
    "renderer" : ["arnold","renderman","vray"],
    "name" : ["pointLight","PxrSphereLight","VRayLightSphereShape"],
    "attribute_intensity" : ["intensity","intensity","intensity"],
    "attribute_color" : ["color","lightColor","color"],
    "attribute_exposure" : ["aiExposure","exposure",""],
    "attribute_visibility" : ["visibility","visibility","visibility"]
},
"directional_light" : {
    "renderer" : ["arnold","renderman","vray"],
    "name" : ["directionalLight","PxrDistantLight","VRaySunShape"],
    "attribute_intensity" : ["intensity","intensity","intensity"],
    "attribute_color" : ["color","lightColor","color"],
    "attribute_exposure" : ["aiExposure","exposure",""],
    "attribute_visibility" : ["visibility","visibility","visibility"]
},
"spot_light" : {
    "renderer" : ["arnold","renderman","vray"],
    "name" : ["spotLight","PxrDiskLight","VRayLightIESShape"],
    "attribute_intensity" : ["intensity","intensity","intensity"],
    "attribute_color" : ["color","lightColor","color"],
    "attribute_exposure" : ["aiExposure","exposure",""],
    "attribute_visibility" : ["visibility","visibility","visibility"]
},
"skyDome_light" : {
    "renderer" : ["arnold","renderman","vray"],
    "name" : ["aiSkyDomeLight","PxrDomeLight","VRayLightDomeShape"],
    "attribute_intensity" : ["intensity","intensity","intensity"],
    "attribute_color" : ["color","lightColorMap","domeTex"],
    "attribute_exposure" : ["aiExposure","exposure",""],
    "attribute_visibility" : ["camera","visibility","visibility"],
    "attribute_shadows" :["aiCastShadows","enableShadows","shadows"]
},
"mesh_light" : {
    "renderer" : ["arnold","renderman","vray"],
    "name" : ["aiMeshLight","PxrMeshLight","VRayLightMesh"],
    "attribute_intensity" : ["intensity","intensity","intensity"],
    "attribute_color" : ["color","lightColor","color"],
    "attribute_exposure" : ["aiExposure","exposure",""],
```

```
            "attribute_visibility" : ["lightVisible","visibility","invisible"]
        }
    }
}
```

materials.json

```
{
    "Materials": {
        "renderer" : ["arnold","renderman","vray"],
        "standard_material": {
            "renderer": ["arnold","renderman","vray"],
            "name": ["aiStandardSurface", "PxrDisneyBsdf", "VRayMtl"],
            "attribute_base":["base","","diffuseColorAmount"],
            "attribute_diffuse_color": ["baseColor", "baseColor", "color"],
            "attribute_metalness": ["metalness", "metallic", "metalness"],
            "attribute_specular":["specular","specularTint","reflectionColorAm
ount"],
            "attribute_specular_color": ["specularColor", "",
"reflectionColor"],
            "attribute_specular_roughness": ["specularRoughness", "roughness",
"roughnessAmount"],
            "attribute_ior":["specularIor","ior","refractionIOR"],
            "attribute_transmission":["transmission","diffTrans",""],
            "attribute_transmission_color":["transmissionColor","transColor","
"],
            "attribute_subsurface":["subsurface","subsurface",""],
            "attribute_subsurface_color":["subsurfaceColor","subsurfaceColor",
""],
            "attribute_subsurface_radius":["subsurfaceRadius","","",""],
            "attribute_subsurface_scale":["subsurfaceScale","","",""],
            "attribute_subsurface_anisotropy":["subsurfaceAnisotropy","","",""],
            "attribute_coat":["coat","clearCoatGloss","coatColorAmount"],
            "attribute_coat_color":["coatColor","clearCoat","coatColor"],
            "attribute_coat_roughness":["coatRoughness","",""],
            "attribute_coat_anisotropy":["coatAnisotropy","",""],
            "attribute_sheen":["sheen","sheenTint","sheenColorAmount"],
            "attribute_sheen_color": ["sheenColor","sheen","sheenColor"],
            "attribute_sheen_roughness":["sheenRoughness","",""],
            "attribute_emission":["emission","",""],
            "attribute_emission_color": ["emissionColor", "emitColor",
"illumColor"],
            "attribute_normal_map": ["normalCamera", "bumpNormal", "bumpMap"]
        },
        "hair_material":{
            "renderer": ["arnold","renderman","vray"],
            "name": ["aiStandardHair", "PxrMarschnerHair", "VRayHairNextMtl"],
```

```
            "attribute_base":["base","diffuseGain","diffuse_amount"],
            "attribute_base_color": ["baseColor", "diffuseColor",
"diffuse_color"],
            "attribute_melanin": ["melanin","","melanin"],
            "attribute_melaninRedness": ["melaninRedness","","pheomelanin"],
            "attribute_melaninRandomize":
["melaninRandomize","","random_melanin"],
            "attribute_roughness": ["roughness","","glossiness"],
            "attribute_ior": ["ior","specularIor","ior"],
            "attribute_specularTint1":
["specularTint","specularGainR","primary_tint"],
            "attribute_specularTint2":
["specular2Tint","specularGainTRT","secondary_tint"],
            "attribute_transmissionTint1":
["transmissionTint","specularGainTT","transmission_tint"],
            "attribute_emission": ["emission","glowGain",""],
            "attribute_emission_color": ["emissionColor","glowColor",""],
            "attribute_opacity": ["opacity","","opacity"]
        }
    }
}
```

6.3 Conversion Workflow

The whole process of conversion starts when the user selects the “From:” and “To:” renderers from the UI and presses the “Convert” button. The setRenderer() function sets the current renderer to the “From:” renderer.

setRenderer()

```
def setRenderer(from_index): #Function to set current renderer
    renderers = ["arnold", "renderman", "vray"]
    if 0 <= from_index < len(renderers):
        cmds.setAttr("defaultRenderGlobals.currentRenderer",
renderers[from_index], type="string")
    else:
        print("Not able to change renderer")
```

After the renderer is set, the respective convert function is called. The convert function first gets the current renderer and checks whether it is available in the dictionary file. It prints an error if it does not find the renderer in the dictionary file. If found, it proceeds with the conversion.

convertLights()

```
def convertLights(fromNumber,convNumber): #Function to convert lights
    lightDictionaryAddress="/transfer/s5512613_SP/Masters_Project/Render_Scene
_Swapper/Dictionary/lights.json"
    with open(lightDictionaryAddress,'r') as file :
        lightsData=json.load(file)
    currentRenderer=getCurrentRenderer()
    #print(currentRenderer)
    check1=checkCurrentRenderer(currentRenderer,lightDictionaryAddress)
    light_conversion_map = {
        lightsData["Lights"]["area_light"]["name"][fromNumber]: "area_light",
        lightsData["Lights"]["point_light"]["name"][fromNumber]:
"point_light",
        lightsData["Lights"]["directional_light"]["name"][fromNumber]:
"directional_light",
        lightsData["Lights"]["spot_light"]["name"][fromNumber]: "spot_light",
        lightsData["Lights"]["skyDome_light"]["name"][fromNumber]:
"skyDome_light",
        lightsData["Lights"]["mesh_light"]["name"][fromNumber]: "mesh_light"
    }
    if check1== "Renderer Supported":
        lightsConversion(convNumber,light_conversion_map,lightDictionaryAddres
s)
    else:
        print("Renderer not supported for conversion")
```

convertMaterials()

```
def convertMaterials(fromNumber,convNumber): #Function to convert materials
    materialDictionaryAddress="/transfer/s5512613_SP/Masters_Project/Render_Sc
ene_Swapper/Dictionary/materials.json"
    with open(materialDictionaryAddress,'r') as file :
        materialsData=json.load(file)
    currentRenderer=getCurrentRenderer()
    #print(currentRenderer)
    check1=checkCurrentRenderer(currentRenderer,materialDictionaryAddress)
    material_conversion_map = {
        materialsData["Materials"]["standard_material"]["name"][fromNumber] :
"standard_material",
        materialsData["Materials"]["hair_material"]["name"][fromNumber] :
"hair_material"
    }
    if check1== "Renderer Supported":
        materialsConversion(convNumber,material_conversion_map,materialDiction
aryAddress)
    else:
        print("Renderer not supported for conversion")
```

The conversion function reads the data from the respective JSON file. Then, it proceeds to check whether any object or light is selected. If something is not selected, it shows an error.

Material Conversion

If selection is found, the conversion function checks whether a shading engine is connected to the selection. If true, it proceeds to determine whether a compatible material is applied to the selection. If true, the copyMaterialAttributes() function is called, and the material attributes are copied to the equivalent material of the other renderer. Otherwise, it checks for displacement and, if found true, connects the displacement with the displacement shader. In the copyMaterialAttributes() function, there is an exception to check for "emission" which are not directly swappable between the two renderers, as Renderman and Arnold have no emission gain attribute. Hence, the emission colour is only copied when the emission gain attribute is not equal to zero.

materialsConversion()

```
def
materialsConversion(convNumber,material_conversion_map,materialDictionaryAddre
ss): # Function to convert material to its equivalent in the other renderer
    with open(materialDictionaryAddress,'r') as file :
        materialsData=json.load(file)
    currentRenderer=getCurrentRenderer()
    number=assignNumber(currentRenderer)
    sel = cmds.ls(sl=True)
    if not sel :
        cmds.confirmDialog(title='Error', message='Please select an object',
button=['OK'], defaultButton='OK')
        cmds.warning("Error : Please select an object")
        return

    for obj in sel:
        objects=cmds.ls(sl=True,dag=True,s=True)
        print ("Selected Objects: ", objects)
        for object in objects:
            print ("Selected Object: ",object)
            shadeEng=cmds.listConnections(object,type='shadingEngine')
            #print(shadeEng)
            if shadeEng :
                for sg in shadeEng :
                    #print (sg)
                    material=cmds.ls(cmds.listConnections(sg),materials =
True)
                    print("Material : ", material)
```

```
                    for mat in material:
                        material_type=cmds.nodeType(mat)
                        #print(material_type)
                        #print("check")
                        if material_type in material_conversion_map :
                            #print("check2")
                            material_type=material_conversion_map[material_typ
e]
                            #print(material_type)
                            #print("check3")
                            shading_group= copyMaterialAttributes(mat,
materialsData, material_type, number, convNumber)
                            cmds.sets(object,edit=True,forceElement=shading_gr
oup)
                        else :
                            if "displacementShader" in material_type:
                                print("mat = ", mat)
                                try:
                                    cmds.connectAttr(mat+'.displacement',shadi
ng_group+'.displacementShader')
                                except:
                                    cmds.confirmDialog(title='Error',
message='Please check the material of your selected object', button=['OK'],
defaultButton='OK')
                                    cmds.warning("Error : Please check the
material of your selected object")
                                    return
                            else:
                                continue
```

copyMaterialAttributes()

```
def copyMaterialAttributes(mat, materialsData, material_type, number,
convNumber): #Function to copy material attributes across different renderers
    convMaterial =
materialsData["Materials"][material_type]["name"][convNumber]
    #print("First Conv Material")
    print("Converting to : ",convMaterial)
    conv= materialsData["Materials"][material_type]["renderer"][convNumber]
    convMaterial = cmds.shadingNode(f"{convMaterial}", asShader=True,
name=f"{mat}_{conv}")
    #print("Second Conv Material")
    #print(convMaterial)
    # Naming wrong of shading group
    convMaterialShadingGroup = cmds.sets(convMaterial, renderable=True,
noSurfaceShader=True, empty=True, name=convMaterial + 'SG')
    #print("Conv material shading group")
```

```
    #print(convMaterialShadingGroup)
    cmds.connectAttr(convMaterial + '.outColor', convMaterialShadingGroup +
'.surfaceShader', force=True)

    components_material_map = {}
    createMaterialComponentMap(components_material_map, materialsData,
material_type, number)
    #print(components_material_map)

    components_convMaterial_map = {}
    createMaterialComponentMap(components_convMaterial_map, materialsData,
material_type, convNumber)
    #print(components_convMaterial_map)

    for attr, convAttr in zip(components_material_map.values(),
components_convMaterial_map.values()):
        #print("entered for loop")
        try:
            print("attr = " , attr)
            print("convAttr = ", convAttr)
            value=cmds.getAttr(mat+'.'+attr)
            print("value = ",value)
            file_node=cmds.connectionInfo(mat+'.'+attr,sourceFromDestination=T
rue)
            file_node=''.join(file_node)
            #print("file_node = ",file_node)
            if "emissionColor" in attr:
                emissiongain=cmds.getAttr(mat+'.emission')
                if emissiongain != 0.0 :
                    # print(emissiongain)
                    # print("entered emission")
                    if file_node != "":
                        #print("entered emission file node")
                        convMaterial=''.join(convMaterial)
                        #print("if file_node is true: convMaterial = ",
convMaterial)
                        cmds.connectAttr(file_node,convMaterial+'.'+convAttr)
                        print("Value set")

                    else:
                        #print("entering emission value set")
                        try:
                            try:
                                cmds.setAttr(convMaterial+'.'+convAttr,value)
                                print("Value set on 1st try")
                            except:
                                cmds.setAttr(convMaterial+'.'+convAttr,value[0
][0],value[0][1],value[0][2],type='double3')
```

```
                                print("Value set on 2nd try")
                            finally:
                                print("Not Set = ", attr)
                        except:
                            if '' in convAttr.lower():
                                continue
                else:
                    continue
            else:
                if file_node != "":
                    convMaterial=''.join(convMaterial)
                    #print("if file_node is true: convMaterial = ",
convMaterial)
                    cmds.connectAttr(file_node,convMaterial+'.'+convAttr)

                else:
                    try:
                        try:
                            cmds.setAttr(convMaterial+'.'+convAttr,value)
                            print("Value set on 1st try")
                        except:
                            cmds.setAttr(convMaterial+'.'+convAttr,value[0][0]
,value[0][1],value[0][2],type='double3')
                            print("Value set on 2nd try")
                        finally:
                            print("Not Set  = ", attr)
                    except:
                        if '' in convAttr.lower():
                            continue
        except:
            pass

    #print("before returning conv material shading group = ")
    #print(convMaterialShadingGroup)
    return convMaterialShadingGroup
```

Light Conversion

If selection is found, the conversion function checks whether it is light. If true, it determines whether a compatible light is a selection. If true, the copyLightAttributes() function is called, and the light attributes are copied to the equivalent light of the other renderer. In the copyLightAttributes() function, there are exceptions to checking and mapping for “exposure” and “color” attributes of the lights. As there is no “exposure” attribute in V-Ray, the intensity changes depending upon the mathematical formula I discussed in my technical background part of the thesis. The “color” attributes of the light are checked for an exceptional case only

when there is a file node connected to it. Renderman and VRay have some different attributes that need to be set before copying the data from the other renderer.

lightsConversion()

```
def lightsConversion(convNumber,light_conversion_map,lightDictionaryAddress):
# Function to convert light to its equivalent in the other renderer
    with open(lightDictionaryAddress,'r') as file :
        lightsData=json.load(file)
    currentRenderer=getCurrentRenderer()
    number=assignNumber(currentRenderer)
    sel = cmds.ls(sl=True)
    if not sel :
        cmds.confirmDialog(title='Error', message='Please select an object',
button=['OK'], defaultButton='OK')
        cmds.warning("Error : Please select an object")
        return

    for obj in sel:
        lights = cmds.ls(sl=True, dag=True, s=True)
        print("Lights : ",lights)
        for light in lights:
            print("Converting Light : ", light)
            light_type = cmds.nodeType(light)
            #print(light_type)
            if light_type in light_conversion_map:
                light_type = light_conversion_map[light_type]
                #print(light_type)
                copyLightAttributes(light, lightsData, light_type, number,
convNumber)
            else:
                continue
```

copyLightAttributes()

```
def copyLightAttributes(light,lightsData,light_type,number,convNumber):
#Function to copy light attributes across different renderers
    convLight=lightsData["Lights"][light_type]["name"][convNumber]
    print("Converting to : ", convLight)
    conv=lightsData["Lights"][light_type]["renderer"][convNumber]
    convLight=cmds.shadingNode(f"{convLight}",asLight=True,name=f"{light}_{con
v}")

    components_light_map={}
    createLightComponentMap(components_light_map,lightsData,light_type,number)
    #print(components_light_map)
```

```
    components_convLight_map={}
    createLightComponentMap(components_convLight_map,lightsData,light_type,con
vNumber)
    #print(components_convLight_map)

    for attr,convAttr in
zip(components_light_map.values(),components_convLight_map.values()):
        try:
            value=cmds.getAttr(light+'.'+attr)
            #print(value)
            file_node=cmds.connectionInfo(light + '.'+ attr,
sourceFromDestination=True)
            #print(file_node)
            file_node=''.join(file_node)
            try:
                if file_node != "":
                    #print("inside file node")
                    if convNumber==1:
                        if number==0:
                            #print("Coming in arnold to renderman_file_node")
                            file_path=cmds.listConnections(light + '.'+
attr,type='file')
                            file_path=''.join(file_path)
                            file_path=cmds.getAttr("%s.fileTextureName" %
file_path)
                            #print(file_path)
                            cmds.setAttr(convLight+'.'+convAttr,file_path,type
='string')
                        else:
                            #print("Coming in vray to renderman_file_node")
                            file_path=cmds.listConnections(light + '.'+
attr,type='file')
                            file_path=''.join(file_path)
                            file_path=cmds.getAttr("%s.fileTextureName" %
file_path)
                            #print(file_path)
                            cmds.setAttr(convLight+'.'+convAttr,file_path,type
='string')

                    if convNumber==2:
                        if number == 0:
                            #print("Coming in arnold to vray_file_node")
                            file_path=cmds.listConnections(light + '.'+
attr,type='file')
                            file_path=''.join(file_path)
                            #print(file_path)
                            cmds.setAttr(convLight+'.useDomeTex',1)
```

```
                            cmds.connectAttr(file_path+'.outColor',convLight+'
.'+convAttr)
                        else:
                            #print("Coming in renderman to vray_file_node")
                            cmds.setAttr(convLight+'.useDomeTex',1)
                            new_file_node=cmds.shadingNode('file',asTexture=Tr
ue)
                            cmds.setAttr(new_file_node+'.fileTextureName',valu
e,type='string')
                            cmds.connectAttr(new_file_node+'.outColor',convLig
ht+'.'+convAttr)

                    if convNumber==0:
                        print("To arnold")
                        if number == 2:
                            #print("Coming in vray to arnold")
                            file_path=cmds.listConnections(light + '.'+
attr,type='file')
                            file_path=''.join(file_path)
                            cmds.connectAttr(file_path+'.outColor',convLight+'
.'+convAttr)
                        else:
                            #print("Coming in renderman to arnold")
                            new_file_node=cmds.shadingNode('file',asTexture=Tr
ue)
                            #print("created new node")
                            cmds.setAttr(new_file_node+'.fileTextureName',valu
e,type='string')
                            cmds.connectAttr(new_file_node+'.outColor',convLig
ht+'.'+convAttr)

                else:
                    try:
                        try:
                            cmds.setAttr(convLight+'.'+convAttr,value)
                        except:
                            cmds.setAttr(convLight+'.'+convAttr,value[0][0],va
lue[0][1],value[0][2],type='double3')
                        finally:
                            pass
                    except:
                        if convNumber==2:
                            if "exposure" in attr.lower():
                                #print("Exposure attribute found. Converting
to vray intensity and value = ", value)
                                intensity_value= cmds.getAttr(light +
'.intensity')
                                if intensity_value !=0:
```

```
                                    #print("intensity_value=
",intensity_value)
                                    new_intensity = intensity_value * (2 **
value)
                                    #print("new_intensity= ",new_intensity)
                                    cmds.setAttr(convLight+'.intensity',new_in
tensity)
                                else:
                                    continue

                            if '' in attr.lower():
                                continue

                    except:
                        if '' in attr.lower():
                            continue
                except:
                    continue

        lightTransform=cmds.listRelatives(light,parent=True,shapes=True,fullPath=T
rue)
        cmds.matchTransform(f"{convLight}", lightTransform)
        lightSet=cmds.listRelatives(lightTransform,parent=True,fullPath=True)
        if lightSet:
            cmds.parent(f"{convLight}", lightSet)
```

When the conversions are complete, the setRenderer() function is again called, changing the current renderer as per the input the user selects. In the end, the conversionComplete() function is called, which prints out a dialogue box showing that the respective conversion has been completed.

conversionComplete()

```
def conversionComplete(convertor): #Function to make a prompt of conversion
complete
    cmds.confirmDialog(title='Conversion Complete', message='Conversion
Complete of all the selected ' f"{convertor}", button=['OK'],
defaultButton='OK')
    cmds.warning("Conversion Complete of all the selected ",convertor)
    return
```

6.4 User Interface Design

The createUI() function designs the simple and user friendly UI which can be used by any level of user. It has background colors and appropriate separators so that the user can distinguish between the two parts of the conversion process. It also mentions the instructions on how to use this tool.

createUI()

```
def createUI(): #Function to create UI and run the code

    def convert_selected_materials(*args): # Function to call the convert
material function
        from_renderer = cmds.optionMenu(from_material_menu, query=True,
value=True)
        to_renderer = cmds.optionMenu(to_material_menu, query=True,
value=True)

        conversion_map = {
            "Arnold": 0,
            "Renderman": 1,
            "VRay": 2
        }

        if from_renderer and to_renderer:
            from_index = conversion_map.get(from_renderer)
            to_index = conversion_map.get(to_renderer)

            if from_index is not None and to_index is not None:
                convertor="Materials"
                setRenderer(from_index)
                convertMaterials(from_index, to_index)
                setRenderer(to_index)
                conversionComplete(convertor)

    def convert_selected_lights(*args): # Function to call the convert
material function
        from_renderer = cmds.optionMenu(from_light_menu, query=True,
value=True)
        to_renderer = cmds.optionMenu(to_light_menu, query=True, value=True)

        conversion_map = {
            "Arnold": 0,
            "Renderman": 1,
            "VRay": 2
        }

        if from_renderer and to_renderer:
            from_index = conversion_map.get(from_renderer)
```

```
            to_index = conversion_map.get(to_renderer)

            if from_index is not None and to_index is not None:
                convertor="Lights"
                setRenderer(from_index)
                convertLights(from_index, to_index)
                setRenderer(to_index)
                conversionComplete(convertor)

    window_name = "Renderer_Scene_Swapper"
    if cmds.window(window_name, exists=True):
        cmds.deleteUI(window_name)

    cmds.window(window_name, title="Renderer Conversion Tool : Maya",
iconName="RCT",widthHeight=(500, 300))

    #cmds.columnLayout(adjustableColumn=True,width=500, height=265)
    #cmds.image(
image='/transfer/s5512613_SP/Masters_Project/Render_Scene_Swapper/Images/logo_
renderer_scene_swapper')
    #cmds.separator(height=100, style='shelf')


    # Creating two optionMenus to choose the conversion from and to renderers
for materials
    cmds.columnLayout(adjustableColumn=True, width=505,
height=265,columnAlign="center")  # Set width and alignment

    cmds.text(label=" Renderer Conversion Tool : Maya ",
height=20,font="boldLabelFont",align="center", backgroundColor=[0.8, 0.8,
0.8])  # Center align with background color
    cmds.text(label=" Note : Please make sure all the objects and lights are
in separate groups. ", height=13,font="smallPlainLabelFont",align="center",
backgroundColor=[0.8, 0.8, 0.8])  # Center align with background color
    #cmds.image(
image='/transfer/s5512613_SP/Masters_Project/Render_Scene_Swapper/Images/logo_
renderer_scene_swapper' )
    cmds.separator(height=10, style='single')

    cmds.columnLayout(adjustableColumn=True, width=500,
columnAlign="center",bgc=[0.8,0.8,0.8])
    # Materials Conversion Section
    cmds.text(label=" Materials Conversion ", height =
25,font="smallBoldLabelFont",align="center", backgroundColor=[0.0, 0.0,
0.3])  # Center align with background color
    cmds.text(label="Please select the object group/groups or all the objects
and then press the CONVERT button:", align="center", backgroundColor=[0.0,
0.0, 0.2])  # Center align
```

```
    from_material_menu = cmds.optionMenu(label="From :", backgroundColor=[0.2,
0.2, 0.2])
    cmds.menuItem(label="Arnold")
    cmds.menuItem(label="Renderman")
    cmds.menuItem(label="VRay")

    to_material_menu = cmds.optionMenu(label="To   :", backgroundColor=[0.2,
0.2, 0.2])
    cmds.menuItem(label="Arnold")
    cmds.menuItem(label="Renderman")
    cmds.menuItem(label="VRay")

    # Convert Button for Materials
    cmds.button(label="Convert", command=convert_selected_materials)

    cmds.columnLayout(adjustableColumn=True, width=500,
columnAlign="center",bgc=[0.8,0.8,0.8])
    # Create a separator
    cmds.separator(height=5, style='shelf')

    cmds.columnLayout(adjustableColumn=True, width=500,
columnAlign="center",bgc=[0.2,0.2,0.5])
    # Lights Conversion Section
    cmds.text(label=" Lights Conversion ", height =
25,font="smallBoldLabelFont",align="center", backgroundColor=[0.3, 0.3,
0.0])  # Center align with background color
    cmds.text(label="Please select the light group/groups or all the lights
and then press the CONVERT button:", align="center", backgroundColor=[0.2,
0.2, 0.0])  # Center align

    from_light_menu = cmds.optionMenu(label="From :", backgroundColor=[0.2,
0.2, 0.2])
    cmds.menuItem(label="Arnold")
    cmds.menuItem(label="Renderman")
    cmds.menuItem(label="VRay")

    to_light_menu = cmds.optionMenu(label="To   :", backgroundColor=[0.2, 0.2,
0.2])
    cmds.menuItem(label="Arnold")
    cmds.menuItem(label="Renderman")
    cmds.menuItem(label="VRay")

    cmds.columnLayout(adjustableColumn=True, width=500,
columnAlign="center",bgc=[0.8,0.8,0.8])
    # Convert Button for Lights
    cmds.button(label="Convert", command=convert_selected_lights)

    cmds.showWindow(window_name)
```

7 Working of Tool

In this chapter, step-by-step screenshots of the tool in action are shown. The screenshots show an Arnold scene being converted to a V-Ray scene. This V-Ray scene is then converted to Renderman. This also shows the flexibility of my tool that it can switch between all the different renderers.

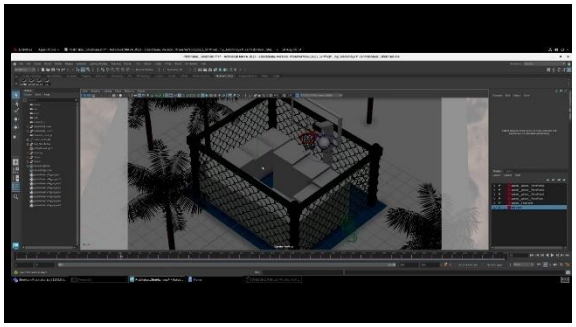
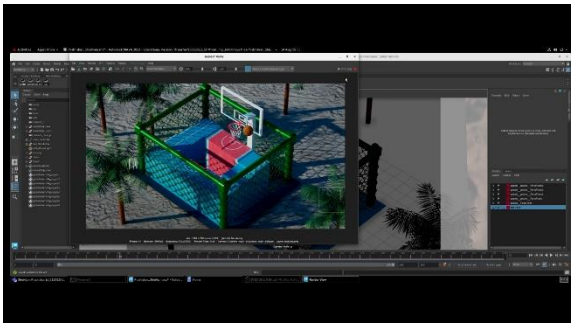

Figure 14 (left side): Showing scene in Autodesk Maya|Figure 15 (right side): Arnold render of scene

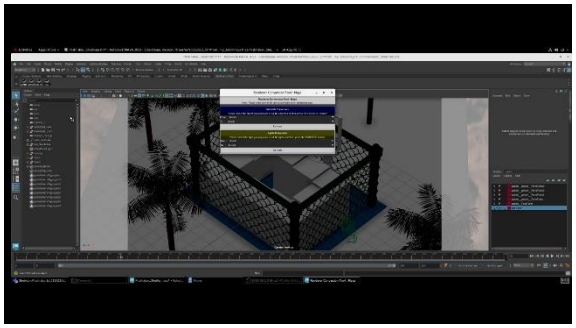
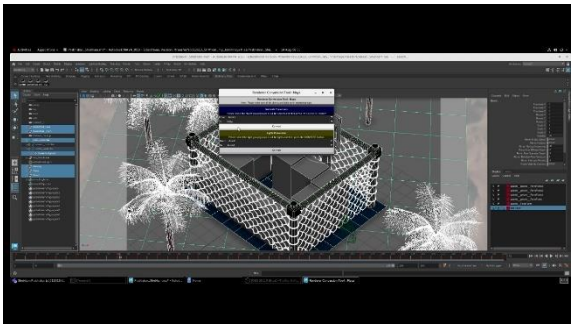

Figure 16 (left side) : Showing UI of tool | Figure 17 (right side): Selecting objects for material conversion

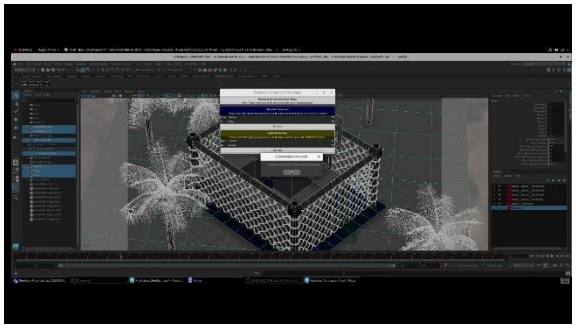
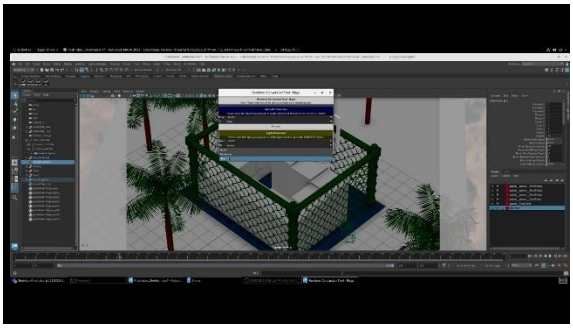

Figure 18 (left side) : Materials converted to V-Ray | Figure 19 (right side) : Selection of lights for conversion

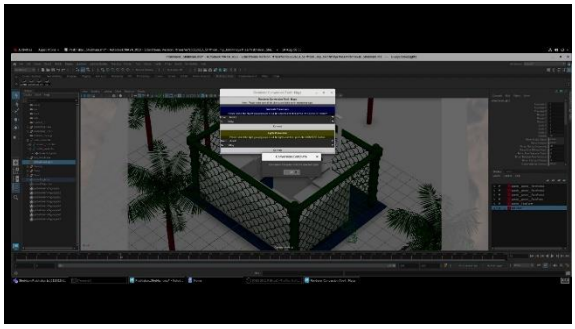
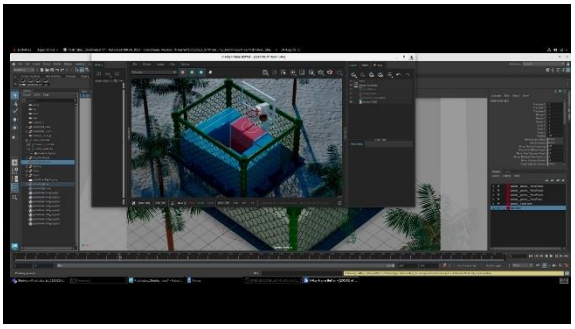

Figure 20 (left side) : Lights conversion complete | Figure 21 (right side) : V-Ray Render of scene

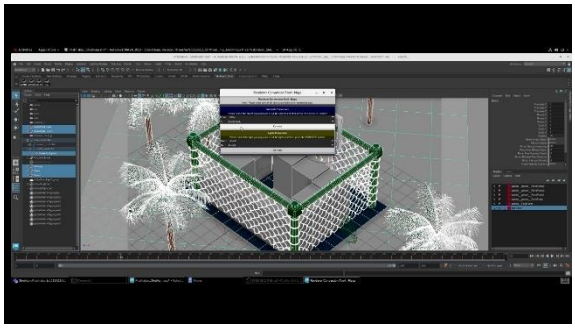
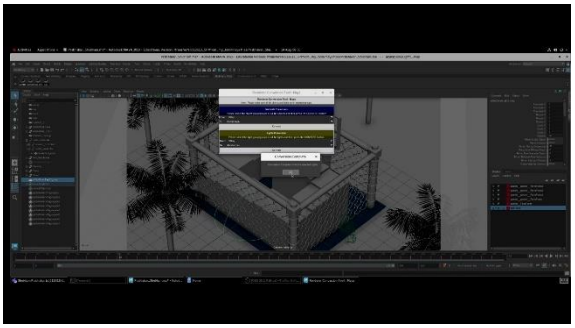

Figure 22 (left side) : Objects converting to Renderman | Figure 23 (right side) : Lights converted

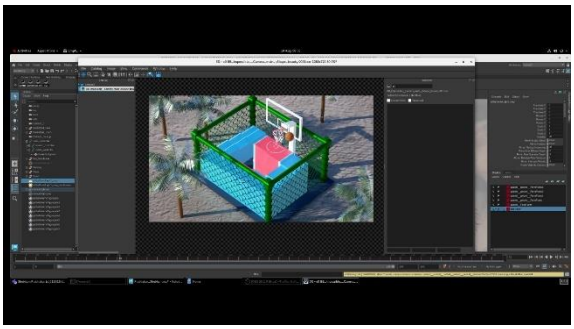

Figure 24 : Renderman render of scene

Figure 25 : Arnold render of scene

Figure 26 : V-Ray render of converted scene

Figure 27 : Renderman render of converted scene

8 Feedback and User Experience

To get feedback about my tool, I spoke to fellow National Centre for Computer Animation (NCCA) students at Bournemouth University. Most of them were my friends and acquaintances. I asked them about my tool's functionality, efficiency and overall user experience.

Firstly, I demonstrated how to use the tool by giving them step-by-step guidance on initially setting it up by copying the required files into the correct folder. I provided them with a ReadMe file so that they understood how to use the tool by themselves.

I have divided the feedback I received from the artists into the following categories.

1. Usability and User Interface: UI was easy to use, and all the instructions were mentioned. The prompt showing completion of conversion after it is complete is quite good. The color scheme of the UI was distracting.

2. Attribute Mapping Accuracy: Most of the attributes were mapped accurately. The textures were also converted and mapped accordingly. The converted scenes were similar to the original, but some inefficiencies existed. In materials, sometimes the converted materials were glossier. The overall look of the scene was sometimes too bright because of the swapping of light attributes between the renderers. Overall, the artist was quite pleased with the look and that they can tweak these attributes quickly now.

3. Efficiency and Speed: The speed of the tool was quite impressive. It converted both the materials and lights in the blink of an eye. Sometimes, when converting to Renderman, it took some time for the textures to be converted to Renderman's native type, the .tex format.

4. Customization and Flexibility: The artists found the tool flexible enough to handle various scenes except some preset values like glass. The inefficiencies found after the conversion could be easily tweaked by changing the attributes in the attribute editor, which they found helpful. The tool saved them quite a sufficient amount of time and tedious work.

5. Areas for Improvement: The color scheme of the UI could be improved. The attributes of the lights had to be tweaked sometimes, which can be improved during the conversion and more work to successfully convert preset values.

Below attached are the renders on which I got the above feedback.

Figure 28, 29 & 30: Arnold to Renderman to V-Ray

Figure 31, 32 & 33: Arnold to V-Ray to Renderman


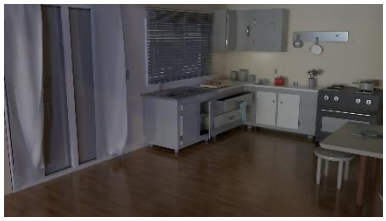


Figure 34, 35 & 36: Arnold to Renderman to V-Ray



Figure 37,38 & 39: Arnold to Renderman to V-Ray



Figure 40, 41 & 42: Arnold to V-Ray to Renderman

9 Conclusion and Future Work

Exploring several rendering engines, creating a flexible Renderer Conversion Tool, and getting insightful input from artists have all contributed to a thorough understanding of the complexities and potentials in the quickly changing field of 3D rendering. This thesis aimed to close the gap between rendering systems, enabling artists to explore different creative directions quickly while maintaining scene integrity and improving workflow effectiveness.

The Renderer Conversion Tool, a product of this research's completion, is evidence of the harmonious fusion of technical advancement and creative expression. The tool's architecture was painstakingly created using an in-depth analysis of Autodesk Maya, Autodesk Arnold, Pixar's RenderMan, and Chaos Group's V-Ray. This allowed the tool to capture each renderer's essence while offering a user-friendly interface that artists of all skill levels can use.

Its use in the artistic community made the tool's revolutionary potential clear. The tool's effectiveness in enabling seamless transitions between rendering engines was confirmed by artist input, which also identified areas that needed improvement. This invaluable feedback will be used to continue my research and development to perfect this tool.

The areas which will be worked upon in the future are:

1. Enhancing the number of compatible renderers.

2. Better automated attribute mapping by writing advanced algorithms or even machine learning models that automate this process.

3. Improving the user interface.

4. Finding ways to integrate my tool in a production pipeline by using Maya standalone.

In the end, I am pleased to say that, in my opinion, my thesis has been a success. I was able to achieve my original goals effectively. The feedback I received will help me further polishing up my tool.

Reference list

Arnold, 2023. *Arnold Hair* [online]. help.autodesk.com. Available from: https://help.autodesk.com/view/ARNOL/ENU/?guid=arnold_core_ac_standard_hair_html [Accessed 18 Jul 2023].

Autodesk, 2021a. *Try Maya Software | Download Maya 2023 - Autodesk* [online]. Autodesk.com. Available from: https://www.autodesk.com/products/maya/free-trial [Accessed 18 Jul 2023].

Autodesk, 2021b. *Arnold Raytracing Renderer Features | Autodesk Official* [online]. Autodesk.com. Available from: https://www.autodesk.com/products/arnold/features [Accessed 18 Jul 2023].

Autodesk, 2023. *Help* [online]. help.autodesk.com. Available from: https://help.autodesk.com/view/MAYAUL/2024/ENU/?guid=GUID-D53B9E3D-E6E3-4CC3-A38F-3AA3A09205E5 [Accessed 18 Jul 2023].

Autodesk, 2023. *Arnold Standard Surface* [online]. help.autodesk.com. Available from: https://help.autodesk.com/view/ARNOL/ENU/?guid=arnold_for_maya_surface_am_Standard_Surface_html [Accessed 18 Jul 2023].

Autodesk, 2023. *Help* [online]. help.autodesk.com. Available from: https://help.autodesk.com/view/ARNOL/ENU/?guid=arnold_user_guide_ac_lights_html [Accessed 19 Jul 2023].

Chaos, 2023. *Lights - V-Ray for Maya - Global Site* [online]. docs.chaos.com. Available from: https://docs.chaos.com/display/VMAYA/Lights [Accessed 20 Jul 2023].

Chaos, 2023. *3D rendering software | Chaos* [online]. www.chaos.com. Available from: https://www.chaos.com/3d-rendering-software [Accessed 20 Jul 2023].

Chaos, 2023. *Materials - V-Ray for Maya - Global Site* [online]. docs.chaos.com. Available from: https://docs.chaos.com/display/VMAYA/Materials [Accessed 19 Jul 2023].

Chaos, 2023. *VRayMtl - V-Ray for Maya - Global Site* [online]. docs.chaos.com. Available from: https://docs.chaos.com/display/VMAYA/VRayMtl [Accessed 20 Jul 2023].

Chaos, 2023. *VRayHairNextMtl - V-Ray for Maya - Global Site* [online]. docs.chaos.com. Available from: https://docs.chaos.com/display/VMAYA/VRayHairNextMtl [Accessed 20 Jul 2023].

Create3dcharacters, 2021. *Maya: Tool Convert Renderer* [online]. Create3dCharacters.com. Available from: https://create3dcharacters.com/maya-tool-convert-renderer/ [Accessed 12 Jul 2023].

lesterbanks, 2020. *Easily Convert from Render Engines in With Maya Scene Converter* [online]. Lesterbanks. Available from: https://lesterbanks.com/2020/07/easily-convert-from-render-engines-in-with-maya-scene-converter/ [Accessed 15 Jul 2023].

mhdmhd, 2021. *Maya Scene Converter* [online]. GitHub. Available from: https://github.com/mhdmhd/MayaSceneConverter [Accessed 15 Jul 2023].

Pixar, 2023. *PxrRectLight* [online]. Renderman Documentation. Available from: https://rmanwiki.pixar.com/display/REN24/PxrRectLight [Accessed 19 Jul 2023].

Pixar, 2023. *Lighting* [online]. Renderman Documentation. Available from: https://rmanwiki.pixar.com/display/REN24/Lighting [Accessed 19 Jul 2023].

Pixar, 2023. *PxrDisneyBsdf* [online]. Renderman Documentation. Available from: https://rmanwiki.pixar.com/display/REN24/PxrDisneyBsdf [Accessed 19 Jul 2023].

Pixar, 2023. *PxrMarschnerHair* [online]. Renderman Documentation. Available from: https://rmanwiki.pixar.com/display/REN24/PxrMarschnerHair [Accessed 19 Jul 2023].

Prabhakar, S., 2023. *Automatic Material Swapper for Maya* [online]. GitHub. Available from: https://github.com/whatshubhamdoes/Automatic_Material_Swapper_Maya [Accessed 20 Jul 2023].

Silke, A., 2021. *Tool Overview – Convert Renderer – Create 3d Characters* [online]. create3dcharacters.com. Available from: https://create3dcharacters.com/2021/07/22/tool-overview-convert-renderer/ [Accessed 12 Jul 2023].