Heloisa Daros dos Santos

ADAPTAÇÃO DA SÉRIE “DIAS VELHO” PARA A RENDERIZAÇÃO NA UNREAL ENGINE 4

Projeto de Conclusão de Curso do Departamento de Expressão Gráfica, Centro de Comunicação e Expressão da Universidade Federal de Santa Catarina, como requisito parcial para a obtenção do grau de bacharel em Design
Orientador: Prof. Me. Flávio Andaló

Florianópolis
2018

Ficha de identificação da obra elaborada pelo autor
através do Programa de Geração Automática da Biblioteca Universitária
da UFSC.

Santos, Heloisa Daros dos
ADAPTAÇÃO DA SÉRIE "DIAS VELHO" PARA A RENDERIZAÇÃO
NA UNREAL ENGINE 4 / Heloisa Daros dos Santos ;
orientador, Flávio Andaló, 2018.
89 p.
Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, , Graduação
em Design, Florianópolis, 2018.
Inclui referências.
1. Design. 2. Rendering. 3. Animação . 4. Animação
3D. I. Andaló, Flávio. II. Universidade Federal de
Santa Catarina. Graduação em Design. III. Título.

Heloisa Daros dos Santos

ADAPTAÇÃO DA SÉRIE "DIAS VELHO" PARA RENDERIZAÇÃO NA UNREAL ENGINE 4

Esta monografia foi julgada adequada para obtenção do Título de "Bacharel em Design", e aprovado a em sua forma final pelo Curso de Design da Universidade Federal de Santa Catarina.

Florianópolis, 13 de junho de 2018.

_____
Profª. Marília Matos Gonçalves, Dr.ª
Coordenadora do Curso

Banca Examinadora:

_____
Prof. Flávio Andaló, Me.
Orientador
Universidade Federal de Santa Catarina

_____
Prof. Gustavo Eggert Boehs, Dr.
Universidade Federal de Santa Catarina

_____
Prof. William Machado de Andrade, Dr.
Universidade Federal de Santa Catarina

Dedico esse trabalho à minha família e aomeu namorado Mateus.

AGRADECIMENTOS

Agradeço aos meus pais, Antônio Carlos e Maria Avelina, que sempre estimularam meu interesse pelas artes e me apoiaram e me escolha profissional, sempre dando o suporte necessário para que conseguisse chegar aqui. Sem seu apoio essa conquista não seria possível.

Aos meus irmãos Daniel e Juliana, pelo companheirismo e sempre estarem lá para me socorrer nos momentos de apuro.

Ao meu namorado Mateus, que junto aos meus familiares, me acompanhou em todos os altos e baixos durante meu trajeto pela universidade e me ajudou a manter a calma em tantos finais de semestre.

Ao meu orientador, professor Flávio, por todo o suporte dado desde os projetos de 3D, sempre dando ideias e tirando dúvidas.

À toda equipe do DesignLab, principalmente ao professor Gustavo por ajudar a resolver vários problemas durante a execução desse projeto e ao amigo André Salomão por me introduzir à Unreal Engine e sempre tirar minhas dúvidas.

Aos professores do Design UFSC, principalmente aos professores responsáveis pelos projetos de animação, que sem os ensinamentos não seria possível concluir este projeto.

Aos amigos que me acompanharam por esta caminhada e a todos que estiveram de alguma forma envolvidos com este projeto.

RESUMO

Este trabalho apresenta as dificuldades encontradas na produção de uma série animada em 3D e propõe uma alternativa a um dos fatores que possui grande influência no atraso da produção: a etapa de rendering. A etapa de rendering é geralmente executada como passo final do processo de produção de uma série animada 3D e consiste em pegar todos os elementos de uma cena 3D e transformá-los em uma imagem final em 2D, nesse caso, os frames que irão compor a sequência de animação final. O processo de rendering pode levar de horas a dias em cada imagem dependendo da complexidade da cena e na capacidade do computador responsável pelo render. Este projeto apresenta o rendering em tempo real através de uma *engine* de jogo como uma solução para acelerar o processo sem perder qualidade de rendering, fazendo uso das ferramentas de *cinematic* disponíveis na *engine* para compor a sequência animada. O produto final desse projeto é adaptação de quatro cenas animadas da série "Dias Velho" para ser renderizada dentro da Unreal Engine 4.

Palavras-chave: Rendering. Animação. Engine de jogos.

ABSTRACT

This work presents the difficulties found on the production of a 3D animated series and proposes an alternative to one of the factors that plays a huge role on production delay: the rendering step. The rendering step is usually performed as the final step of the production process of a 3D animated series and consists on taking all the elements of a 3D scene and transforming them on a final 2D image, on this case, the frames that are going to compose the final animation sequence. The rendering process can take from hours to days on each image depending on the complexity of scene and the capacity of the computer responsible for the render. This project presents real time rendering through a game engine as a solution to speed up the process without losing rendering quality, making use of the cinematic tools available on the engine to compose the animated sequence. This project's final product is the adaptation of four animated scenes from the "Dias Velho" series to be rendered inside Unreal Engine 4.

Keywords: Rendering. Animation. Game Engine.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

2D – Bidimensional
3D – Tridimensional
UE – Unreal Engine
R&D – Research & Development (Pesquisa e Desenvolvimento)
VFX – Visual Effects (Efeitos Visuais)
CPU – Central Processing Unit
GPU – Graphics Processing Unit
GI– Global Ilumination (Iluminação Global)
UE Unreal Engine.
PPV Post Process Volume.

SUMÁRIO

1 INTRODUÇÃO

1.1 APRESENTAÇÃO DO TEMA

Produzir uma animação é uma verdadeira corrida contra o tempo, que envolve uma série de prazos a serem cumpridos. Em um projeto para televisão, onde diversos episódios precisam ser realizados em um curto período de tempo e com orçamento reduzido, o desafio é ainda maior.

Catherine Winder e Zahra Dowlatabadi, autoras do livro *Producing Animation* (2011), utilizaram como analogia a relação existente entre a velocidade da produção de uma série com a de um corredor da prova de sprint no atletismo. Segundo elas, nos dois casos, tanto o atleta quanto a série de televisão precisam percorrer uma distância no menor tempo possível. Independentemente da quantidade de episódios, se faz necessário que a alta velocidade seja mantida durante todo o processo de produção da série, para que a mesma seja bem-sucedida.

A primeira série de animação totalmente produzida em gráficos computadorizados em 3D foi *Reboot* em 1994. Desde então, diversas séries que utilizaram a mesma técnica e foram lançadas, obtendo variados níveis de sucesso. Entre as séries bem sucedidas encontramos títulos como *Jimmy Neutron* (2002) e mais recentemente a série *Miraculous Ladybug* (2016).

Figura 1 Imagem da série Reboot.

Fonte: Mainframe Entertainment.

Em uma produção de animação 3D para televisão é difícil alcançar os mesmos padrões de alta qualidade de materiais, texturas e iluminações produzidas por filmes de grandes estúdios. Um dos principais empecilhos está nas máquinas de renderização gráfica, pois não há tempo que estas renderizem em qualidade semelhante. Tornando o processo de renderização um dos maiores desafios do emprego da técnica em animações para séries televisivas.

Renderizar é tornar os gráficos tridimensionais que são trabalhados no computador, em uma imagem 2D. É como se uma foto daquele objeto 3D fosse tirada. O processo de renderização é o que irá produzir cada *pixel* de uma foto, para que possamos visualizá-la. Um segundo de animação geralmente gira em torno de 24 frames, ou seja, vinte e quatro dessas "fotos" precisam ser tiradas.

O processo para que cada uma dessas imagens seja formada é algo que pode levar alguns minutos ou até dias, dependendo da complexidade visual da cena e dos elementos que a constroem como: modelos, texturas, tipos de materiais e iluminação. O tempo que se leva para a formação de cada uma dessas imagens é o que chamamos no contexto de produção de tempo de rendering.

Levando-se em conta que, o tempo limitado é uma das grandes dificuldades enfrentadas pelas produções televisivas, constata-se que existe uma grande dificuldade de se renderizar algo tão complexo num curto período de tempo. Também, por consequência dos menores orçamentos encontrados nessas produções, nem sempre há recursos para se investir em mais ou em melhores máquinas, o que acaba fazendo com que o produtor opte por simplificar a parte artística do projeto, de forma a conseguir cumprir os prazos estabelecidos.

É nesse contexto, que as *engines* de jogo, que trabalham com renderização em tempo real, surgem como uma alternativa interessante. Dependendo do porte do projeto é possível conseguir um render de melhor qualidade a um menor custo computacional para a máquina. Essa ferramenta que foi criada para rodar jogos com elementos complexos em tempo real já que foi otimizada para isso.

A possibilidade de utilizar esses programas para a animação vem através das ferramentas para criação de *cinematics*, os curtos filmes de animação que são vistos dentro dos jogos, que essas *engines* possuem. A Unreal Engine, que pertence à empresa Epic Games, tem particularmente incentivado produtores de animação a usarem sua *engine* para essa finalidade.

Figura 2- Imagem de curta renderizado na Unreal Engine

Fonte: Epic Games

Em 2015, a Epic Games anunciou no blog oficial da Unreal Engine 4 que esta seria oferecida gratuitamente para download para quem possuísse o interesse em aprendê-la e utilizá-la comercialmente. Para a utilização da *engine* em projetos de vídeo não se faz necessário o pagamento de royalties à empresa (ao contrário da produção de jogos cuja uma taxa por jogo vendido é cobrada).

Com todo esse incentivo, aos poucos, estúdios estão implementando o uso de *engines* de jogos no auxílio de produções animadas 3D para televisão. Um exemplo disso é o estúdio Digital Dimension que está produzindo a série *Zafari* (ainda sem previsão de lançamento), primeira série de animação com destino à televisão a ser totalmente renderizada dentro da Unreal Engine.

Figura 3 – Imagem da série Zafari

Fonte: Digital Dimension.

A partir desse panorama, esse projeto tem como intuito explanar as potencialidades e as dificuldades de se utilizar a ferramenta de renderização Unreal Engine 4, aplicando esse conhecimento na releitura

de algumas cenas da série “Dias Velho”, série com propósito educativo voltada para o público infantil que está sendo produzida pelo DesignLab, laboratório vinculado ao curso de Design da Universidade Federal de Santa Catarina.

Ao longo dos capítulos, abordaremos os objetivos do projeto, seus problemas e serão apresentados estudos sobre rendering, elementos a serem levados em consideração durante a sua execução e a relação desses estudos com a UE (Unreal Engine 4), que é a ferramenta que se pretende utilizar no projeto.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Adaptação de cenas de uma série animada 3D para renderização em tempo real.

1.2.2 Objetivos Específicos.

- Aprofundar conhecimentos sobre rendering através de pesquisa em livros e sites sobre o assunto.
- Aprofundar conhecimentos sobre a UE através do estudo da documentação disponível.
- Avaliar a melhor forma de exportar os modelos e animações produzidas para a série e executar o processo.
- Adaptar os materiais previamente configurados VRay no editor de materiais da UE.
- Adaptar quatro cenas da série “Dias Velho” para renderização na UE
- Gerar uma saída de vídeo final.

1.3 JUSTIFICATIVA

Animação produzida em 3D é a principal técnica aplicada por grandes estúdios de animação, como a Pixar em produções de longa-metragem. Entretanto, estas produções são onerosas tanto no aspecto financeiro quanto no tempo de produção. O minuto animado em estúdio de grande porte como Pixar ou Dreamworks custa em média 1,5 milhão de dólares (WRIGHT, 2015).

Em produções televisivas, além do menor orçamento (em média 13 mil dólares o minuto animado), os prazos mais curtos também afetam a produção. As animações digitais em 2D acabam adaptando-se melhor a essas limitações, por isso é uma técnica que segue ainda forte na produção televisiva e em pequenas produções independentes.

Um dos elementos que influenciam no tempo de uma produção 3D é a fase de rendering, Josh Prykril, diretor de computação gráfica da série

televisiva *Butt-Ugly Martians* (2001) declara que o "tempo de rendering, o equivalente técnico ao *'cel painting'* nos antigos dias do 2D, pode ser o tudo ou nada em uma produção de animação para TV mesmo que todo o resto dê certo".-

Figura 4 Josh Prykril e personagens de Butt-Ugly Martians

Fonte: Animation World Network.

Analisando esses fatores, o projeto propõe a utilização de uma *engine* de jogo (no caso a Unreal Engine 4) para a renderização em tempo real de uma série de televisão sob a justificativa de tornar o processo mais rápido e entregar uma animação de melhor qualidade visual, sem afetar o orçamento.

David Dozoretz, criador da série Zafari, declara que "a Unreal nos possibilita chegar a uma aparência próxima a Pixar com orçamentos e velocidades de televisão". Ele também explica que a nova velocidade de produção possibilitou que sua equipe começasse a desenvolver determinadas etapas, como a iluminação das cenas, que geralmente ficariam mais para o final, já nos primeiros passos do processo de produção da animação.

1.4 DELIMITAÇÕES DO PROJETO

Levando-se em conta as limitações de tempo, decidiu-se que o projeto focaria apenas em: transferência dos modelos estáticos e animados para a Unreal Engine produzidos pela equipe do DesignLab para a animação "Dias Velho"; conversão dos materiais do VRay para materiais da engine, considerando a possibilidade de refinar alguns materiais caso seja possível; importação da animações para a UE; montagem de algumas cenas da animação dentro da ferramenta de produção de sequências animadas disponível na *engine* e exportação dessas cenas para uma saída de vídeo final.

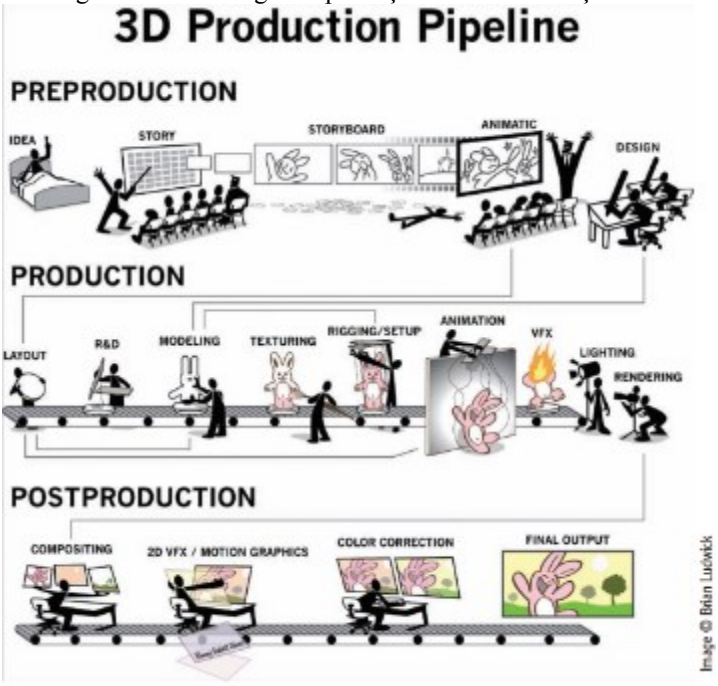
O projeto limita-se apenas em mexer nos materiais originais já produzidos pela a equipe laboratório para eventuais correções de problemas de exportação e importação, sem a intenção de refazer modelos ou animações, tampouco criar novos.

2 METODOLOGIA PROJETUAL

Dado que o projeto estará envolvido em um contexto de produção de uma série de animação 3D, escolheu-se uma metodologia apresentada no livro *3D Animation Essentials* (2012) de Andy Beane. A metodologia é divida em três etapas principais: pré-produção, produção e pós-produção, com cada uma sendo subdivididas em etapas menores.

Os processos executados neste projeto estarão principalmente inseridos nas etapas de produção e pós-produção, não estando envolvido o desenvolvimento de nenhum processo referente à pré-produção, entretanto, esta etapa também será explanada no próximo tópico, visto que seu conhecimento é relevante ao projeto.

Figura 5 Metodologia de produção de uma animação 3D.



Fonte: BEANE, 2012.

2.1.1 Pré-produção

Como o nome sugere, é o momento que vem antes da produção, onde será feita toda a preparação para a mesma. É nessa etapa em que a proposta visual do projeto e do design dos personagens, cenários e objetos começam a ser desenvolvidos. Uma pré-produção bem trabalhada é essencial para o sucesso de qualquer projeto de animação.

Referem-se a pré-produção:

- **Ideia/História:** Da ideia inicial vem à história, que é trabalhada e transformada em narrativa, que nesse ponto é apenas um apanhado geral do total da história. Esse é o momento em que os personagens, detalhes e o conflito principal da história são esboçados.
- **Script:** É o escrito final da história e possui uma formatação específica. Descreve o que o espectador irá ver e ouvir na tela, tendo escrito nele os movimentos básicos do personagem, cenário, tempo, ações e diálogo.
- **Storyboard:** É a representação visual do script, semelhante a uma história em quadrinhos. Nesse estágio começa-se a dar uma maior atenção aos planos de câmera, também é onde são decididas as principais poses dos personagens nas cenas e alguns efeitos visuais.
- **Animatic:** Versão animada do storyboard. Uma animação simples e limitada, porém já adequada ao timing pretendido na animação final, é utilizado para ver como a história está fluindo e se algo precisa ser modificado.
- **Design:** É etapa onde a aparência final do projeto é decidida, inclui design de personagens, objetos, vestimentas e cenário de acordo com as necessidades do projeto.

A pré-produção de todo o material referente ao episódio piloto da série já havia sido produzida no DesignLab, essa documentação foi analisada pela aluna para saber qual era o universo da história, seus personagens e eventos que aconteciam, avaliou-se quais cenas seriam adaptadas através do *storyboard* e qual era a estética da série de forma a realizar uma adaptação coerente com o trabalho de design previamente desenvolvido.

2.1.2 Produção

Encaminhadas as etapas principais da pré-produção, começa-se a efetivamente produzir o projeto. A metodologia compreende que as etapas que fazem parte desse processo são:

- **Layout:** Versão 3D do *animatic* previamente feito em 2D. Pode começar na pré-produção e se estender até a pós. Importante para ver como determinados ângulos de câmera, distância e escala entre objetos e personagens funciona em um ambiente 3D. O layout serve como guia para os outros artistas e animadores que trabalharão nas próximas etapas.

- **Pesquisa e Desenvolvimento (R&D):** A sigla R&D vem do inglês Research & Development, que pode ser traduzido para o português como pesquisa e desenvolvimento. Essa é outra etapa que pode permear todo o processo de produção. A equipe responsável pela pesquisa e desenvolvimento deve trabalhar para superar limitações técnicas que os artistas e animadores podem vir a encontrar no desenvolvimento do projeto.
- **Modelagem:** Um modelo é a representação geométrica de uma superfície de um objeto que pode ser visualizada e rotacionada em um ambiente 3D. Existem diferentes programas e técnicas para criação de modelos, um exemplo é a técnica de escultura digital que é utilizada em programas como o ZBrush da Pixologic e o Mudbox da Autodesk.
- **Texturização**: É onde os artistas configuraram materiais e texturas dos modelos criados. Assim como na modelagem, essa etapa possui diferentes técnicas e programas dedicados a isso, cabe à equipe de produção definir qual se adequa melhor às limitações do projeto.
- **Rigging:** O esqueleto do personagem é adicionado controles ao modelo de forma a possibilitar que os animadores o movam.
- **Animação:** Etapa onde o movimento dos objetos e personagens é criado.
- **Efeitos visuais 3D (VFX):** É onde a animação de tudo que não sejam personagens e os objetos com os quais eles podem vir a interagir é feita. Inclui, por exemplo, animações de componentes como: cabelo, pelagem, tecido, água, fogo, poeira, entre outros.
- **Iluminação:** Com a ajuda os guias de cor criados na pré-produção, os artistas iluminam a cena, ajudando a dar o tom à cena.
- **Rendering:** Etapa que converte a imagem 3D em uma imagem 2D.

Este projeto lidará com as etapas de: pesquisa e desenvolvimento, efeitos visuais, iluminação, *rigging* e rendering. Os materiais referentes às outras etapas virão do material já previamente produzido pelo laboratório, como os modelos dos personagens e seus respectivos *rigs*, elementos de cenário, texturas e animações, estes irão ser importados para a UE, fazendo as adaptações necessárias de forma que essa importação seja bem sucedida.

2.1.3 Pós-produção

Finalização do projeto, onde serão feitos os refinamentos finais. Essa etapa também pode ser usada para corrigir problemas apresentados em algumas cenas da animação. A metodologia define como etapas da pós-produção:

- ***Compositing*:** Junção de todos os elementos visuais produzidos para gerar a saída final de vídeo.
- **Efeitos visuais 2D/*Motion Graphics*:** Geralmente misturam-se ao estágio de *compositing*, podendo ser feito pelo mesmo artista. Irá trabalhar efeitos que são mais simples de serem trabalhados no final do projeto 2D do que durante a produção 3D.
- **Correção de cor:** Ajuste do projeto para que a cor mantenha-se consistente durante toda a animação e que esteja de acordo com a proposta artística do projeto.
- **Saída final**: A saída final da animação pode ser em forma de: filme, vídeo, internet.

As etapas de pós-produção a serem trabalhadas nesse projeto serão: *compositing*, correção de cor e saída final. Estes e os outros processos correspondentes ao projeto previamente citados serão explicados em detalhes no capítulo 4.

3 COLETA DE DADOS E ANÁLISE

Nesse capítulo serão reunidas as informações que serão à base da teórica do projeto. No primeiro tópico será explicada um pouco sobre a engine que será adotada para a renderização do projeto e como ela é utilizada na criação de filmes. Do segundo tópico serão apresentados alguns conceitos básicos de rendering, métodos e outros assuntos relevantes ao tema rendering. Finalmente, o terceiro tópico apresentará em mais detalhes a série de animação "Dias Velho".

3.1 UNREAL ENGINE

A Unreal Engine foi criada pelo fundador da Epic Games, Tim Sweeney. Seu desenvolvimento começou em 1995, tendo sido desenvolvida para o jogo de tiro em primeira pessoa chamado *Unreal,* que foi lançado ao mercado em 1998. A versão utilizada no jogo é chamada Unreal Engine 1, a versão mais atual disponível no mercado é a Unreal Engine 4, geralmente abreviada como UE4 ou apenas UE.

Figura 6 Captura de tela da Unreal Engine 1

Fonte: Epic Games.

A UE foi à primeira *engine* 3D para computador a permitir edições em tempo real, possibilitando ao artista adicionar elementos em cena como, por exemplo, luzes e saber como ficaria no jogo.

A possibilidade de criação e edição de *cinematics*[1] dentro da Unreal Engine foi implementada numa segunda versão através da ferramenta chamada Matinee, que foi utilizada pela Epic Games na *cinematic* de introdução do jogo *Unreal Tournament 2003*. A atual ferramenta para

[1] *Cinematics*, no contexto de jogos eletrônicos, são curtas sequências de animação não interativas que aparecem no decorrer do jogo, geralmente introduzindo elementos a narrativa.

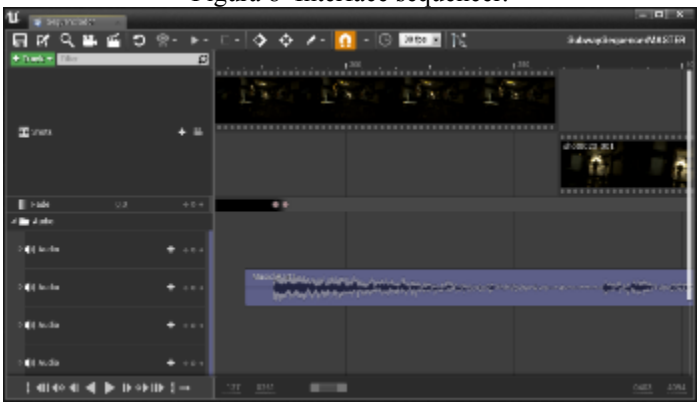
criação de *cinematics* chama-se Sequencer e foi introduzida em 2016 na versão 4.12 da UE4 como sucessora do Matinee.

Figura 7 Introdução do jogo Unreal Tournament 2003.

Fonte: Epic Games.

Dentro do Sequencer é possível configurar câmeras (com a possibilidade de movimentá-las na cena) para gravar sequências de animação. A interface da ferramenta trabalha com uma linha do tempo semelhante à de outros programas de edição de vídeo como o Adobe Premiere, onde os clipes que compõem as cenas podem ser cortados e ordenados da forma como o artista desejar, sendo possível também adicionar áudio à sequência criada.

Figura 8 Interface sequencer.



Fonte: Autora.

É possível visualizar os *cinematics* dentro da própria *engine*, através da *cinematic viewport* ou então através do botão *play,* que no contexto de desenvolvimento dos jogos ativa um modo onde é possível visualizar o jogo rodando, mas que também pode ser configurado para rodar a sequência quando clicado.

Figura 9 Botão play na interface da engine.

Fonte: Autora.

3.2 RENDERING

Como explicado anteriormente, no capítulo 2, o processo de rendering é o último estágio da produção de uma animação 3D. Renderizar significa pegar todos os modelos 3D, *rigs*, animações, materiais, texturas, efeitos visuais 3D e transformá-los em uma imagem 2D (AKENINE-MÖLLER et al., 2008).

3.2.1 Rendering off-line e rendering em tempo real

Rendering off-line é o tipo de rendering comumente usado em produções de animação 3D tanto em longas, quanto em curtas e produções para televisão, sendo também utilizado em outras áreas como: efeitos visuais, visualização arquitetônica, entre outras.

No rendering off-line podem-se levar horas ou até dias para gerar uma única imagem ou *frame* de animação, por isso, grande parte das produções faz uso de *render farms*, isto é, uma rede de computadores conectados entre si dedicados apenas ao processo de renderização.

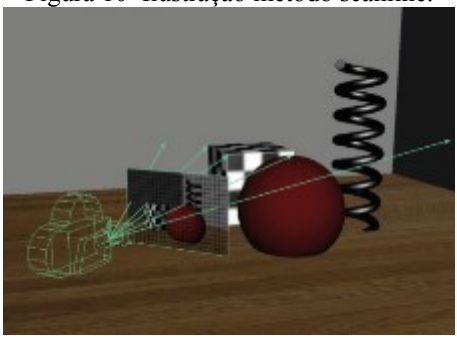
Alguns programas considerados renderizadores *off-line* encontrados na indústria são: Mental Ray da NVIDIA, Vray do Chaos Group, Renderman da Pixar, Arnold da Solid Angle, entre outros.

O autor Andy Beane consta que a maioria dos renderizadores tradicionais renderizam suas imagens através de duas técnicas básicas: *scanline* e *raytracing*.

O método *scanline* pode ser representado por uma série de linhas horizontais que sai de uma câmera virtual captando informações dos elementos da cena que interceptam a linha para transformar essas informações em imagem.

Possui a vantagem de ser muito rápido, porém não é capaz de calcular refração, reflexão nem iluminações globais mais complexas. Ótimo método para ser utilizado na etapa de pré-visualização, podendo também ser utilizado como render final em projetos que não demandem desses cálculos mais complexos que o *scanline* não é capaz de performar.

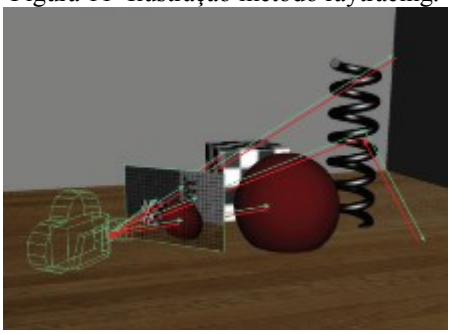
Figura 10 Ilustração método scanline.



Fonte: BEANE, 2012.

O método de *raytracing* é mais completo que o *scanline*, porém ao custo de exigir mais do computador. Baseia-se em iluminação global, traçando raios a partir da câmera até e os testando nos objetos da cena, caso o objeto seja reflexivo o raio irá rebater até encontrar um objeto não reflexivo, quando os raios pararem eles retornam para câmera com a informação encontrada no processo (AKENINE-MÖLLER et al., 2008).

Figura 11 Ilustração método raytracing.



Fonte: BEANE, 2012.

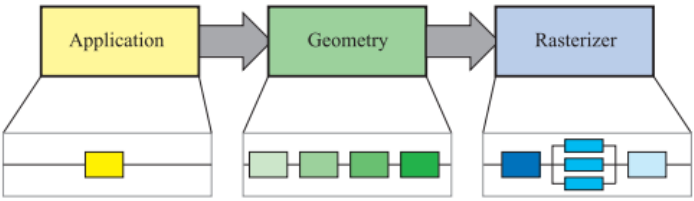
O rendering em tempo real está diretamente relacionado a geração de imagens rápidas que possibilitem ao observador imergir no ambiente virtual, dando ao espectador um senso de conexão com o espaço tridimensional (AKENINE-MÖLLER et al., 2008). O uso GPUs é muito discutido nas literaturas referentes a esse tipo de rendering por suas possibilidades na otimização desse processo.

Comercialmente é mais utilizado na área de games, estando presente em *engines* como a Unreal Engine, Unity 3D, CryEngine, entre outras.

Quando se fala de rendering em tempo real um conceito frequentemente encontrado na literatura a respeito é a *graphics rendering pipeline*, considerada como base do rendering em tempo real.

Esse modelo explica etapa por etapa como acontece o processo de rendering, dividindo-o em três principais estágios aplicação, geometria e *rasterizer*, sendo representado pelo fluxograma abaixo.

Figura 12 Gráfico ilustrando a graphics rendering pipeline



Fonte: AKENINE-MÖLLER et al, 2008.

O estágio da aplicação, como o próprio nome sugere, é conduzido pela aplicação sendo executado pela CPU,

No estágio da geometria é computado o que da imagem deve ser renderizado, como e onde. É o estágio que lida com transformações, projeções, cálculos que dizem respeito à aparência do objeto (como materiais e interações de luz), executa o processo de *clipping*[2] e transforma as coordenadas.

O *rasterizer* utiliza toda a informação gerada nos estágios anteriores para então, efetivamente, gerar a imagem, fazendo o objetivo geral de este estágio ser designar as cores exatas aos pixels de forma que a imagem seja gerada corretamente.

.

3.2.2 Estilos de rendering

3.2.2.1 Foto-realista

Como o próprio nome descreve, é um estilo de rendering cujo objetivo é gerar uma imagem ou vídeo, onde os materiais, texturas e iluminação dos elementos presentes em cena sejam os mais próximos possíveis da realidade, assemelhando-se a uma fotografia. É um método muito utilizado em simulações arquitetônicas para mostrar ao cliente uma aproximação do resultado final do projeto.

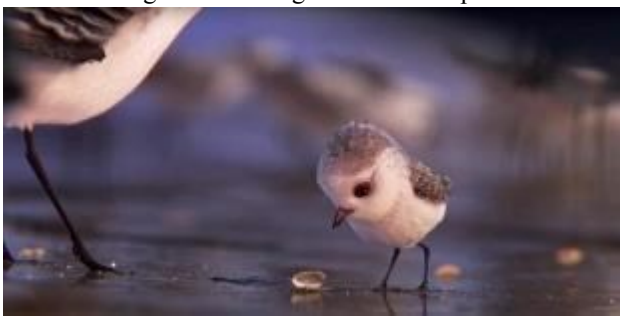
[2] As geometrias fora da vista da câmera não são renderizadas, sendo descartadas. O processo de *clipping* trabalha com as geometrias que se encontram na intersecção entre espaço dentro e fora da vista, descartando apenas a informação que não se encontra na vista.

Figura 13 Cena feita no 3D max e renderizada no VRay.

Fonte: Estúdio Render Collective.

Entretanto, o fotorrealismo não é unicamente utilizado em simulações, podendo também ser aplicado em animações 3D. O curta-metragem *Piper* (2016) produzido pelo estúdio Pixar é um exemplo de projeto de animação que faz uso de um estilo de rendering foto-realista. O design dos personagens do curta é estilizado, mas os materiais, texturas e iluminação foram trabalhados para que se aproximassem da realidade.

Figura 14 – Imagem do curta Piper.



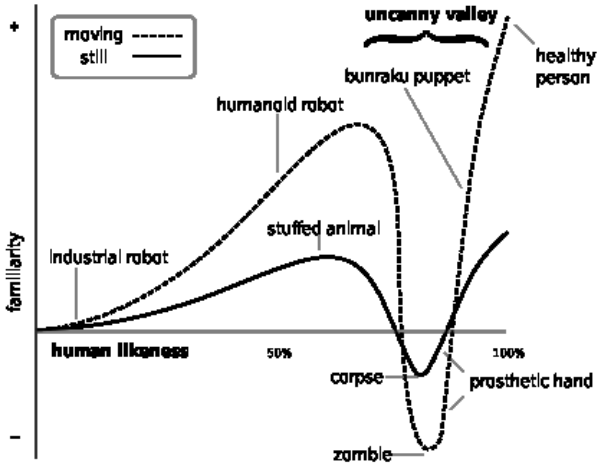
Fonte: Pixar.

A técnica de fotorrealismo encontra problemas quando aplicada a personagens humanos, devido a um conceito chamado de *vale da estranheza*, que vem originalmente da robótica, mas também é aplicado a personagens 3D e bonecos com aparência humana.

A teoria explica que até um determinado nível, um robô com características humanas nos causa empatia. Contudo, quanto mais ele se assemelha a aparência humana, maior a chance de ele cair em um vale onde nos causa estranheza, pois ele é quase humano, mas claramente não é um ser vivo, já que as características que os tornam não humano nos chamam muito a atenção.

Esse efeito se amplifica quando o robô ou personagem movimenta-se, em decorrência de ser muito difícil replicar os movimentos e expressões humanas de forma natural. Podendo nos dar a impressão de que o robô ou personagem é um cadáver se mexendo, o que acaba ocasionando um sentimento de desconforto ainda maior. Alguns filmes conhecidos possuírem personagens que caem no vale da estranheza é *Final Fantasy: Spirits Within* (2001) e *Expresso Polar* (2004).

Figura 15 - Representação gráfica da teoria do vale da estranheza

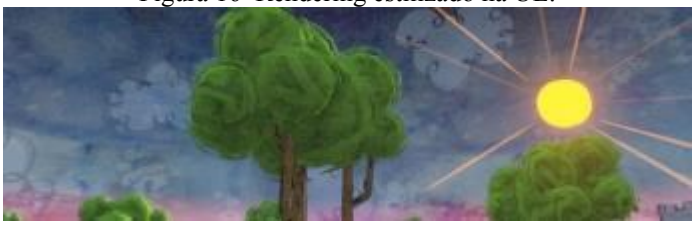


Fonte: BEANE, 2012.

3.2.2.2 Não foto realista

O rendering não foto-realista é um tipo de rendering estilizado, ao contrário do foto-realista não possui um objetivo claro, vindo a depender do projeto. Por exemplo, o objetivo de alguns métodos de rendering não foto-realista é o de criar imagens similares a ilustrações técnicas (AKENINE-MÖLLER et al., 2008). Existem também diversos estudos em rendering não foto-realista de simulação de técnicas de pintura e outras mídias tradicionais.

Figura 16 Rendering estilizado na UE.



Fonte: Documentação Unreal Engine 4.

Uma das técnicas mais populares de rendering não foto-realista utilizados na área de animação chama-se *toon shading* também conhecido pelo nome de *cel shading* cujo objetivo é simular imagens similares as que seriam produzidas em uma animação 2D.

Muitas vezes é o *toon shading*, ou *cel shading*, é integrado na produção de projetos em 2D para criação de cenas que seriam muito complexas ou demoradas de serem produzidas em 2D. Como por exemplo, cenas que demandam complexos movimentos de câmera. Um exemplo de animação 2D que integra elementos 3D em *toon shading* é o longa de animação japonesa *Your Name* (2016), que utilizou a técnica na composição de algumas cenas através do plugin para 3Ds Max PSOFT Pencil+.

Figura 17 PSOFT Pencil+ funcionando dentro do 3Ds Max

Fonte: Cartoon Brew.

3.3 TÓPICOS PERTINENTES AO PROJETO

Os tópicos presentes nessa seção, que apesar de serem consideradas áreas de estudo próprias, aparecem de forma recorrente em literaturas sobre rendering, por serem considerados fatores que influenciam no tempo de rendering, tornando seu conhecimento importante para desenvolvimento do projeto.

3.3.1 Texturização

"Texturização é o processo de criar as superfícies e atributos de cor dos modelos e fazê-los parecer com qualquer objeto que eles devem supostamente representar" (BEANE, 2012). O processo de texturizar um modelo envolve configurar as UVs do objeto, definir o material e pintar as texturas que sejam necessárias.

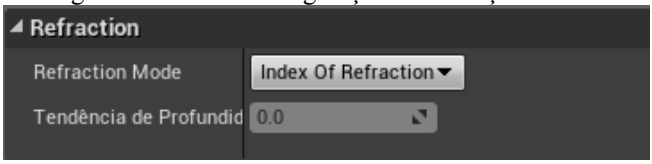
UVs são um mapa de coordenadas do espaço em 2D aplicada a um objeto 3D, U significa horizontal e V vertical. É como se a superfície do elemento fosse planificada de um plano 2D. Do mapeamento das UVs é gerado um *UV map* que será utilizado pelo artista para, por exemplo, pintar as texturas do modelo em um software como Adobe Photoshop.

Figura 18 Modelo de objeto 3D e seu mapa UV.

Fonte: Documentação Unreal Engine.

Com os UVs configurados o artista irá designar materiais ao modelo. Esses materiais possuem atributos configuráveis que serão utilizados para simular as propriedades daquele material: alguns desses atributos que se encontram na maioria dos renderizadores são: cor difusa, transparência, reflexão, refração, translucência, entre outros.

Figura 19 Aba de configurações de refração na UE.



Fonte: Autora.

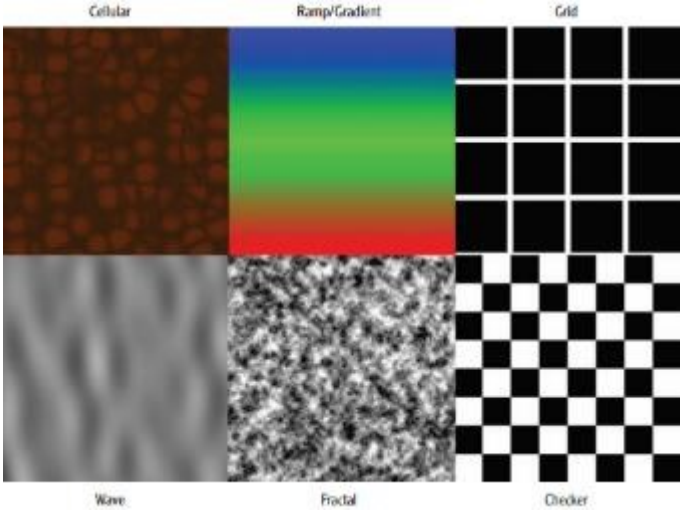
O artista responsável pela texturização precisa ter uma visão apurada, para não apenas ser capaz de reproduzir as cores e propriedades dos materiais da mesma forma que se comportariam no mundo real, como também estar atento a detalhes. Um exemplo seria perceber se o material estará sujo ou desgastado, já que nenhum material é totalmente perfeito na vida real.

Imagine, por exemplo, uma história onde os personagens se encontram em uma casa abandonada. Os móveis provavelmente estarão envelhecidos ou desgastados, é importante prestar a atenção nesses detalhes, pois as texturas e materiais dos elementos de cena também ajudam a ambientar a história.

Para adicionar esses detalhes extras o artista de texturização faz uso dos chamados mapas de textura, que são configurados dentro do material que o artista está trabalhando. Existem dois tipos de mapas de textura: os procedurais e os baseados em imagens bitmap.

Os mapas de textura procedurais possuem a vantagem de não dependerem de resolução, sendo possível aumentá-los sem causar distorções na imagem ou perda de qualidade da mesma. Também são *seamless*, isto é, não apresentam marcas entre as áreas que conectam as texturas. Esses tipos de mapas são úteis para grandes áreas aonde uma mesma textura irá se repetir várias vezes. Alguns exemplos de mapas procedurais podem ser vistos na figura abaixo.

Figura 20 Exemplos de mapas de textura procedurais.



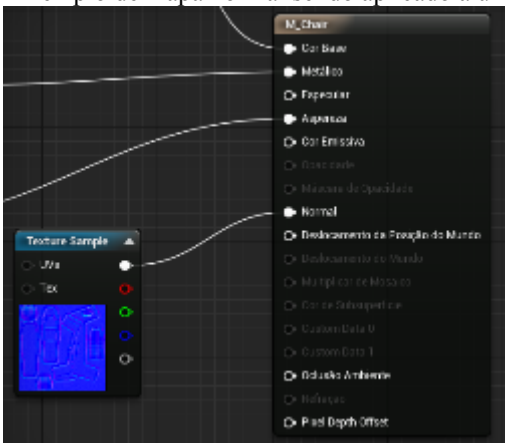
Fonte: BEANE, 2012.

Os mapas de textura gerados por imagens bitmap geralmente são editados em algum editor de imagens como o Adobe Photoshop. Antes de serem trazidos para dentro do programa, são úteis quando o artista quer texturizar uma área específica do modelo ou criar texturas mais realistas a partir fotos. Ao contrário dos mapas procedurais, os mapas de bitmap dependem da resolução, então haverá distorção caso a imagem seja aumentada.

Algumas propriedades dos materiais que podem ser controladas por texturas são: cor difusa, *bump*, *specular*, transparência, reflexão, *displacement* e *normal*.

O artista deve ter consciência de que determinados mapas consomem mais recursos de rendering do que outros e optar pelo que funcionará melhor para o projeto. Por exemplo, se o artista quer que a textura do objeto dê a impressão de ser levemente afundada em determinadas áreas, ele pode usar um mapa de *bump* ou *normal* que ajudará a simular esse efeito ao invés de um mapa de *displacement* que simularia de forma ainda mais realista, já que ele altera o formato dos polígonos do modelo. Contudo, consumiria muito mais tempo de rendering.

Figura 21 Exemplo de mapa normal sendo aplicado a um material.



Fonte: Autora.

Existem várias técnicas para o artista pintar as texturas no modelo: ela pode ser pintada à mão em um software como Adobe Photoshop, criada através da manipulação de fotos ou pintada diretamente em cima do modelo 3D em programas como o Autodesk Mudbox ou o Zbrush da Pixologic.

Dentro da UE, materiais e texturas podem ser configurados através do *material editor* que é uma interface gráfica que trabalha com um sistema de *nodes*, isto é, os materiais são configurados através de "nós" que conectam os elementos que os compõem.

Figura 22 Interface do material editor na UE.

Fonte: Autora.

3.3.2 Iluminação

Como explicado na metodologia de produção 3D apresentada no capítulo 2, a iluminação ajuda a dar tom à cena. De acordo com BEANE (2012) profissionais de iluminação são similares aos artistas que trabalham em iluminações de filmes e pintores artísticos, pois é importante que eles tenham um bom conhecimento de como a luz se comporta no mundo real para sejam capazes de simularem dentro do programa.

No processo de iluminação, primeiramente o artista decide com que tipo de luz trabalhar. Cada tipo de luz se comporta de uma forma e possui um tempo de render diferente, por isso é importante que o artista as conheça para saber qual será mais apropriada para o projeto em que está trabalhando.

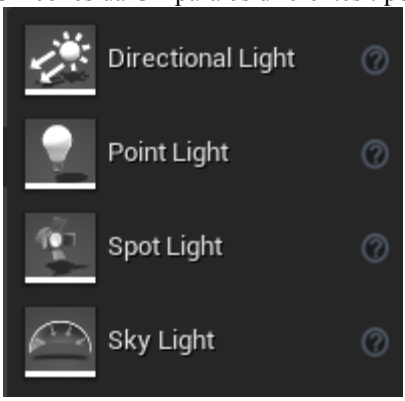
Segundo BEANE (2012) alguns dos tipos mais comuns de luzes encontrado em programas de 3D são:

- **Spotlight:** Luz sai de um único ponto e direção, assemelha-se a um cone.
- **Luz omni:** Semelhante a uma lâmpada, luz sai de um único ponto em todas as direções, chamada de *point light* em alguns programas.
- **Luz direcional:** Luz sai em raios paralelos, assim como a luz do sol.
- **Luz ambiente:** Cria luz em volta de si com intensidade semelhante em todos os ângulos.
- **Luz de área:** Emite luz de uma área ou superfície.

Cada um desses tipos de luz possuem atributos que podem ser manipulados para que se aproximem ao tipo de iluminação desejado, alguns desses atributos são: intensidade, cor, atenuação e sombra.

A documentação da UE define que os tipos de luzes encontrados na engine são: spotlight, luz direcional, *point light* e *sky light* (tipo de iluminação que simula a luz do céu).

Figura 23 Ícones da UE para os diferentes tipos de luz.



Fonte:Autora.

Na aba de configuração da luz encontra-se uma aba de transformação similar, encontrada em todos os *actors*[3], ou atores de cena, onde se permite que sejam configuradas sua localização no espaço, rotação e escala.

Há também a opção de escolher a *mobility* (mobilidade, em português) da luz. Cada um dos tipos de *mobility* na UE afeta o desempenho do rendering de uma determinada forma, por isso é importante ter noção desse conceito, entre os tipos de mobilidade encontram-se:

- ***Static*:** Como o nome sugere, são luzes estáticas, não podendo ser movidas enquanto a sequência ou jogo está rodando. São calculadas dentro de *lightmaps*[4] e depois de processadas pela *engine*, não impactam mais no desempenho, sendo o tipo de mobilidade com menor custo para a máquina. Não podem

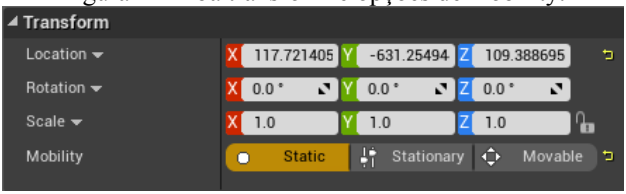
[3] Nomenclatura genérica usada pela UE para qualquer objeto que possa ser adicionado a cena (modelos, câmeras, luzes, etc) que seja capaz de sofrer alterações em sua translação, rotação e escala.

[4] Texturas que contém informações de luz pré-computadas para acelerar o processo de rendering.

sombrear objetos dinâmicos (objetos que são movidos pelo personagem).

- ***Stationary*:** Luzes do tipo *stationary* são feitas para permanecerem em uma posição. Entretanto é possível configurá-las para que performem algumas mudanças durante a sequência como, por exemplo, mudanças de cor e brilho, tornando essa a maior diferença entre ela e a luz do tipo *static*, porém vale notar que essas diferenças apenas afetam as luzes diretas, com as luzes indiretas sendo configuradas em *lightmaps*. Não pode ser movida, rotacionada e nem ter sua área de influência modificada durante a sequência.
- **Movable:** O tipo mais dinâmico, porém também com maior custo para a máquina, pode mudar quase todas as suas características durante a sequência, entretanto não possui suporte para luzes indiretas em tempo real.

Figura 24 Aba transform e opções de mobility.



Fonte: Autora.

Cada um dos tipos de luz disponíveis na UE (citados anteriormente) possuem também diversos tipos de configurações próprias que podem ser modificadas de forma a conseguir os efeitos desejados, sem grande comprometimento no desempenho.

3.3.3 Iluminação Global (GI)

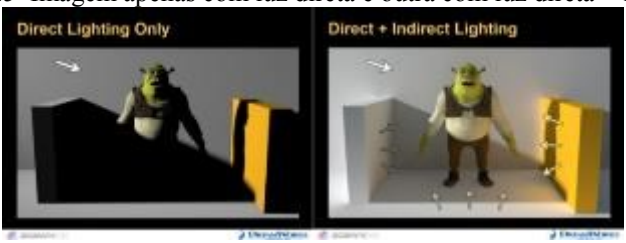
A iluminação global, também comumente chamada de GI (do inglês *Global Illumination*) é um termo geral para descrever um grupo de algorítimos avançados para o cálculo de iluminação e materiais (BEANE, 2012). Esses algorítimos concentram-se no cálculo da luz indireta, isto é, a luz que rebatida de uma superfície para outras. A luz direta seria a luz que sai da fonte luminosa e bate diretamente na superfície.

Esses cálculos acrescentam um grau de realismo à cena, deixando a luz mais natural e permitindo efeitos como o *color bleeding*[5], por isso

[5] Efeito de transferência de cor entre objetos próximos em uma cena, causa pela reflexão colorida da luz indireta (BIRN, 2002).

GI é um assunto muito discutido, principalmente, na área de rendering foto-realista.

Figura 25 Imagem apenas com luz direta e outra com luz direta + indireta.



Fonte: Dreamworks.

Essa ferramenta está diretamente relacionada ao aumento do tempo total de rendering, pois esses cálculos consomem muitos recursos da máquina para ser computado, o que também dificulta sua aplicação em ferramentas de rendering em tempo real. Atualmente ainda não é possível obter efeitos de GI em qualidade semelhante à de renderizadores *off-line* em uma interface que trabalhe em tempo real como, por exemplo, um vídeo game.

A UE atualmente não possui uma ferramenta que permita um efeito de GI dinâmico, possuindo ferramentas que apenas trabalham com iluminação pré-computada através de *lightmaps* como o *Lightmass Importance Volume*.

3.4 A SÉRIE DE ANIMAÇÃO "DIAS VELHO"

Dias Velho é um projeto de uma série animada em 3D educativa cujo público-alvo são crianças em idade pré-escolar. O projeto pertence ao DesignLab, laboratório vinculado ao curso de Design da Universidade Federal de Santa Catarina.

A série é uma adaptação na *graphic novel* (história em quadrinhos de caráter não seriado) "Dias Velho e os Corsários" de Eleutério Nicolau da Conceição, publicada em 1988, sendo uma das várias histórias em quadrinhos com o foco na história de Santa Catarina publicadas pelo autor.

A história se passa no século XVII e tem como personagem principal Francisco Dias Velho, bandeirante paulista que veio a Santa Catarina e foi responsável pelo processo de colonização da atual cidade de Florianópolis. A trama tenta ser um relato histórico da época baseando-se em documentos oficiais porém também introduzindo alguns eventos fabricados por falta de material disponível para consulta. A história original foca-se no conflito entre Francisco Dias Velho e o corsário

Thomas Frins, que acaba por resultar no assassinato de Dias Velho pelo corsário, um elemento que evidencia o caráter não seriado da história.

A adaptação feita pelo DesignLab propôs introduzir eventos fictícios da vida cotidiana nesse contexto histórico, adaptando a linguagem para um público mais jovem e um formato de narração episódico. O conflito maior entre Frins e Dias Velho nunca ocorre na história da série, outros acontecimentos da *graphic novel* original, que envolvem temas incompatíveis com o público pré-escolar, como violência e assédio sexual, também foram retirados.

Novos personagens principais foram criados de forma a conduzir a história da adaptação, esses novos personagens são as crianças Leca e Cauã, com o macaco sagui falante Gui sendo companheiro de ambas.

Leca é uma menina fruto da miscigenação dos povos brasileiro, corajosa e impulsiva, possui seis anos. Cauã um índio criado por jesuítas, introvertido e reflexivo, possuindo sete anos, ambos representam habitantes da ilha relevantes ao período histórico tendo o Gui como representante da fauna local. Os personagens infantis visam gerar empatia com o público-alvo. Na adaptação, Dias Velho foi delegado a um papel de mentor das crianças e Thomas Frins transformado em um vilão caricaturado, que comanda uma trupe de piratas atrapalhados, adicionando humor à história.

Figura 26 Concept art da casa de Dias Velho.

Fonte DesignLab.

Figura 27 Dias Velho caricaturado, ao centro, com as crianças e Gui.

Fonte: DesignLab.

O projeto atualmente encontra-se em fase de produção do episódio piloto, já possuindo a parte de pré-produção referente a este episódio concluída. O recorte escolhido para ser desenvolvido neste PCC, foram as quatro primeiras cenas da primeira sequência do episódio piloto, devido ao fato cenas mais avançadas em sua produção, logo possuíam todos os elementos necessários para fazer uma adaptação completa para a Unreal Engine.

4 DESENVOLVIMENTO

Nesse capítulo será descrito o processo de adaptação da série Dias Velho para a UE. Como citado brevemente no capítulo referente à metodologia do projeto, a animação já havia passado pela fase de pré-produção do episódio piloto, encontrando-se na produção do mesmo.

A parte de produção estava parcialmente desenvolvida com todos os personagens e cenários modelados, mapas de UVW configurados, texturas pintadas, *rigging* dos personagens feitos e algumas animações prontas. O renderizador que estava sendo utilizado no projeto até então era o VRay, todos os materiais haviam sido configurados para ele.

Pode-se dividir o desenvolvimento desse projeto em cinco etapas:

- Exportação dos personagens e elementos de cenário do 3Ds Max e importação para a UE.
- Configuração de materiais dentro da UE com base nos materiais desenvolvidos no VRay.
- Exportação das animações em fbx e alembic para serem utilizadas dentro da *engine*.
- Montagem das cenas dentro da ferramenta Sequencer.
- Pós-produção: utilizar ferramentas de pós-produção disponibilizadas na *engine* para fazer eventuais refinamentos e gerar uma saída de vídeo final.

Vale constar que as versões dos programas utilizadas no desenvolvimento desse projeto foram 3Ds Max 2018 e Unreal Engine 4.19, visto que podem vir a serem encontradas diferenças em algumas ferramentas entre diversas versões de um mesmo programa.

4.1 EXPORTAÇÃO DE PERSONAGENS E CENÁRIO DO 3DSMAX PARA A UE

4.1.1 Personagens

Dois personagens foram importados para a montagem das cenas: Leca e Cauã, as crianças principais da série. Essa seção irá descrever como ambos foram importados de forma suas malhas e hierarquias de *bones* fossem importadas corretamente para dentro da *engine* de forma que pudesse receber animações.

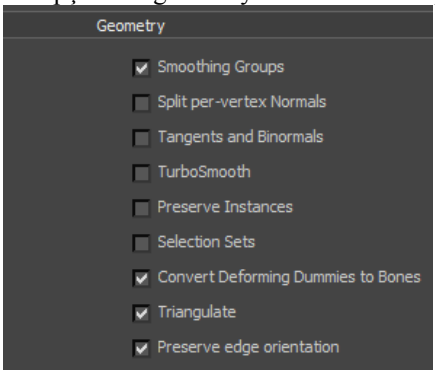
Os personagens foram exportados do 3Ds Max no formato fbx. Esse formato de arquivo é proprietário à Autodesk, sendo a principal forma de se trocar informações entre os programas de 3D pertencentes à empresa. Alguns programas de outras empresas também possuem suporte para importação de arquivos fbx como é o caso da UE.

O fbx é um formato popular para trabalhos em 3D, pois eles podem conter em si: geometria dos modelos, hierarquia de *bones,*[6] deformadores como *skin*[7] e *morph targets*[8], materiais, animações, câmeras e luzes, fazendo-o um grande facilitador na troca de informação entre diferentes programas.

A UE é um dos programas que aceita a importação de arquivos fbx, logo, o formato foi utilizado no projeto na transferência do material que fora produzido para a série dos programas de 3D para a UE, sendo a única exceção as animações do rosto, como será discutido no tópico referente à exportação de animações.

O tópico *FBX Best Practices* (UNREAL ENGINE, 2015) encontrado dentro da documentação da UE sugere algumas opções a serem ativadas na exportação do fbx para que ele vá corretamente para *engine*. Nas opções que dizem respeito à geometria do objeto é sugerido marcar as opções: *smoothing groups*, *preserve edge orientation* e deixar desmarcada a opção *tangentes e binormals*. Nesse projeto, além dessas opções também foi marcada a opção *convert deforming dummies to bones*.

Figura 28 Opções de geometry selecionadas no projeto.



Fonte: Autora.

No 3Ds Max, *smoothing groups* são um grupo numerados de 1 a 32 e podem ser designados a áreas de um objeto. Se duas faces de uma ponta compartilham o mesmo número de *smoothing group* a borda entre as duas faces é considerada suave, de outro modo é considerada facetada.

---

[6] "Ossos" que são usados para controlar modelos animados.

[7] Deformador que permite que determinados conjuntos de vértices do modelo sejam designados a um *bone*, sofrendo influência do mesmo.

[8] Versões deformadas de uma mesma malha, utilizadas para animação através da interpolação entre as mesmas.

Esses números podem ser configurados através do *edit poly* ou então adicionando um modificador chamado *smooth*.

Quando se trabalha com *engines* de jogos é desejável manter o personagem com o menor número de polígonos possível de forma a não afetar a performance ao renderizar, os *smoothing groups* são muito úteis pois ajudam a suavizar a malha sem alterar o número total de polígonos, ao contrário de outros métodos que usam subdivisões de polígonos para suavizar a malha.

A desvantagem é que a suavização oferecida pelos *smoothing groups* não é tão refinada quanto a de métodos que fazem uso de subdivisões como *turbosmooth* do 3Ds Max, porém existem outras possíveis configurações para ajudar a suavizar a malha dentro da *engine* como o *tessellate*, que será discutido no tópico de configuração de materiais.

No projeto, todos os modificadores de *turbosmooth* dos personagens e do cenário foram removidos e trocados por modificadores de *smooth* com seus *smoothing groups* sendo exportados via fbx.

Além das três opções sugeridas pela página de *FBX Best Practices* (UNREAL ENGINE, 2015), a opção *convert deforming dummies to bones* acabou sendo incluída, pois o *rig* dos personagens foi configurado através do CAT (Character Animation Tool). Esse *rig* possui uma geometria que geralmente acaba atravessando a malha do personagem para ajudar em sua configuração, entretanto, no 3Ds Max ela pode ser facilmente escondida para evitar aparecer no render final. Ao ser importada para a *engine*, essa geometria funde-se a malha caso essa opção não esteja marcada como, por exemplo, na imagem abaixo:

Figura 29 Exemplo do erro descrito.

Fonte: Autora

Na imagem podemos ver a geometria do CAT atravessando os dedos, no 3dsmax isso poderia ser facilmente escondido e não atrapalharia o modelo do personagem, porém na UE acaba unindo-se a malha, causando problemas.

Apenas as opções acimas não são suficientes para que o personagem venha com seu *rig* e *skin* funcionando, também foi necessário marcar a opção *animation* e dentro dela a opção *deformations* e *skin*.

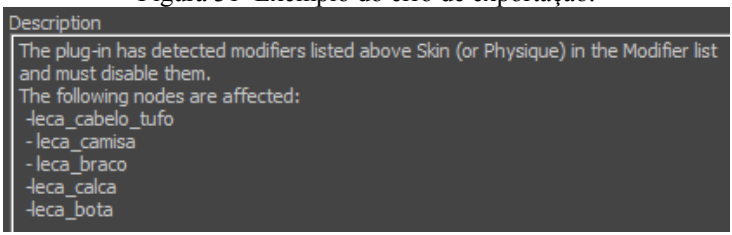
Figura 30 Configuração aba deformations.

Fonte: Autora.

Todos os personagens da série possuem *morph targets*, utilizados em suas expressões faciais, esses *morphs* são organizados através de um script para 3Ds Max para facilitar o trabalho dos animadores da série. Porém, para esse projeto não foi necessário que eles fossem exportados devido ao uso do formato alembic na exportação das animações faciais, o que será mais bem explicado no tópico referente a exportação dessas animações, logo, a opção *morphs* não precisou ser marcada.

Observou-se também que, nas partes do personagem que possuíam o deformador *skin*, ele deveria ser o modificador no topo da lista de modificadores do elemento, caso contrário, o modificador acima do *skin* seria descartado. Por exemplo, se o *smooth* estiver acima do *skin* os *smoothing groups* não seriam exportados corretamente. O exportador do fbx envia uma mensagem de aviso nessas situações.

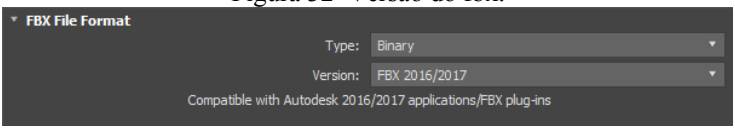
Figura 31 Exemplo do erro de exportação.



Fonte: Autora.

A versão do fbx recomendada pela documentação da UE é a 2016, entretanto não se verificou sérios problemas de importação da personagem para dentro da UE em decorrência da utilização de uma versão diferente. No projeto utilizou-se a versão 2016/2017 do 3Ds Max 2018.

Figura 32 Versão do fbx.

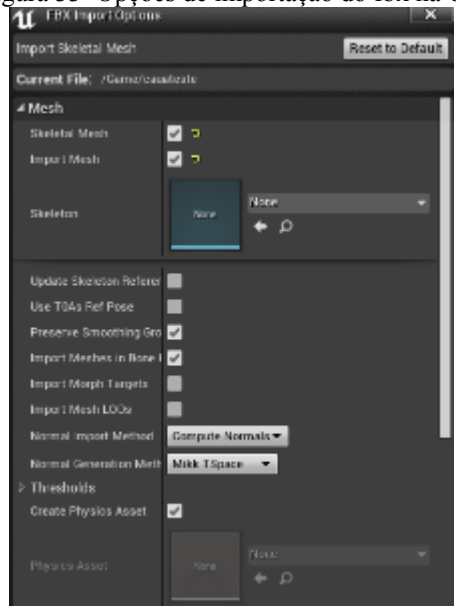


Fonte: Autora.

Com essas opções marcadas, o fbx já pode ser salvo e importado para dentro da *engine*. Os personagens foram importados como *skeletal mesh*, que é a nomenclatura usada pela UE para malhas com *skin*, isto é, malhas vinculadas a uma hierarquia de *bones*.

Personagens animáveis sempre serão importados como *skeletal meshes* para a UE, porém não é apenas utilizado para personagens já que alguns objetos de cena podem possuir *bones* para auxiliar em sua animação.

Figura 33 Opções de importação do fbx na UE



Fonte: Autora.

Após a importação, são encontrados três tipos de *assets*[9] relacionados ao modelo, um denominado como sendo a própria *skeletal mesh*, seu *skeleton* separado e seu *physics asset*. O *skeleton* separado é o *asset* ao qual a animação foi relacionada após ser importada para dentro

[9] Nomeclatura comumente usada em documentações para *engines* de jogos refere-se aos elementos que compõem um jogo.

da *engine* e o *physics asset* é que lidará com a parte de física e colisões do modelo. Ao clicar na *skeletal mesh* encontram-se os materiais que compõem o personagem e ver sua hierarquia de *bones*.

Figura 34 Menu skeletal mesh.

Fonte:Autora.

Duas versões dos personagens foram importadas para dentro da *engine*: uma com seu corpo completo e a outra sem cabeça. A versão com a cabeça serviu para configurações de materiais e conferir se os olhos não atravessavam a malha, a versão sem a cabeça foi utilizada para a montagem das cenas na ferramenta Sequencer, como será explicado com maior detalhe em tópicos mais adiante.

Figura 35 Assets da personagem Leca.

Fonte: Autora.

4.1.1.1 Problemas verificados durante a importação

Durante a preparação do personagem é importante se certificar de que ele está com as escalas e normais corretas. Como o projeto lidou com dois elementos diferentes (corpo e cabeça separados), com as escalas incorretas pode ser difícil fazer com que as duas partes se encaixem quando forem trabalhadas juntas. No caso das normais, elas podem estar invertidas fazendo com que alguns elementos do modelo ou texturas apareçam de forma incorreta após a importação.

Figura 36 Erro nas normais invertidas e correção.

Fonte: Autora.

No projeto foi necessário corrigir as escalas do corpo do personagem Cauã e algumas normais que estavam invertidas (aro do óculos, pé e braço esquerdos).

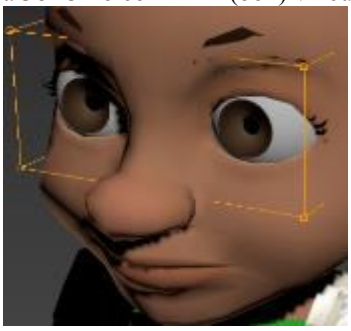
Observou-se também que o olho da personagem Leca estava saindo levemente para fora da malha quando importado para dentro da *engine*, o que causaria problemas quando ela fechasse os olhos na animação. Foi verificado que o que estava causando o problema era um modificador de FFD (*Free Form Modifier*) que estava influenciando no formato do olho, como os modificadores de FFD não são exportados pelo fbx, a informação era perdida e ao ser importada para dentro da UE era possível ver o olho levemente para fora.

Figura 37 Exemplo do olho atravessando a malha.

Fonte: Autora.

A solução foi colapsar na lista de modificadores o modificador responsável pela relação do FFD com formato do olho, assim, integrando as modificações ao modelo, fazendo que mantivesse o tamanho desejado. O modificador em questão era o *FFD Binding* que estava fazendo a relação da forma do olho com um FFD(box) 2x2x2.

Figura 38 Olho com FFD (box) vinculado.



Fonte:Autora.

Também é necessário prestar atenção se a *skin* do personagem está se comportando corretamente, pois durante uma das importações percebeu-se que o *skin* da personagem Leca estava apresentando problemas quando uma animação era aplicada ao modelo, causando erros na forma como o personagem deformava ao se mover. É possível testar isso apenas mexendo nos *bones* do personagem no painel de *skeletal mesh* ou então aplicando uma animação a ele.

Avaliou-se que o erro foi provavelmente causado durante o processo de preparação da personagem para a importação da UE, a solução foi refazer a preparação do zero e reimportar a personagem.

Figura 39 Problema de skin.

Fonte: Autora.

Figura 40-Problema corrigido: bota não deforma

Fonte: Autora.

4.1.2 Cenário

O cenário escolhido para ser importado para dentro da *engine* foi a casa do personagem Dias Velho, o primeiro cenário a aparecer na série. Cada objeto do cenário foi exportado separadamente e importado para dentro da UE.

A exportação dos elementos do cenário também foi feita em fbx com configurações semelhantes a dos personagens, apenas não marcando a opção *convert deforming dummies to bones* na aba *geometry* e não marcando nenhuma opção na aba *animation*.

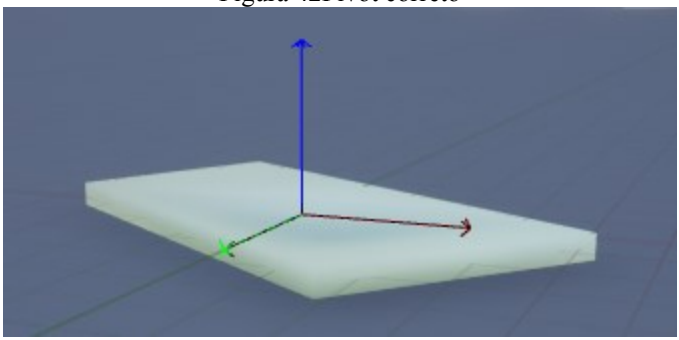
A principal preocupação dessa etapa foi ter certeza de que cada elemento havia sido alinhado com o ponto zero (ponto onde todas as três coordenadas encontram-se no zero) do 3Ds Max antes que os objetos fossem exportados, caso contrário o *pivot* do objeto desloca alinhando-se

ao ponto zero da UE ao invés de alinhar-se ao modelo da forma como fora configurado dentro do programa de 3D.

Figura 41 Pivot errado.

Fonte:Autora.

Figura 42Pivot correto



Fonte:Autora.

É possível automatizar alguns processos do 3dsmax através do uso da linguagem *maxscript*[10], existem várias opções online de scripts disponibilizados de graça que ajudam no processo de centralizar cada elemento na cena e exportá-los para fbx, um dos scripts utilizados nessa etapa para auxiliar o processo foi o Batch Import/Export de Jon Balcaen disponibilizado no site ScriptSpot (acessado em abril de 2018).

Assim como nos personagens, todos os modificadores de *turbosmooth* foram excluídos e substituídos por *smoothing groups* nos objetos onde era necessário suavizar a malha em algum ponto, como, por exemplo, nos arbustos, árvores, etc.

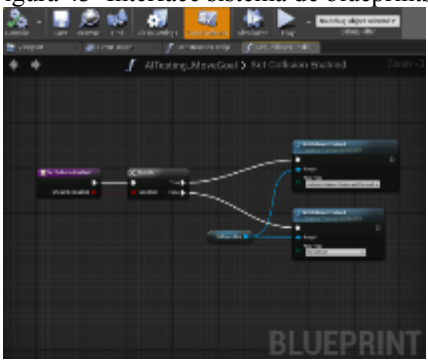
O processo de importação para a UE também é semelhante ao executado na etapa anterior, entretanto, como os elementos de cenário serão em sua maioria estáticos eles então foram importados como *static mesh*[11], porém, apesar de não possuírem *bones* a *engine* ainda oferece a possibilidade de tornar esses elementos animáveis através do sistema de *blueprints*. O sistema de *blueprints* da UE é um sistema de programação

[10] Linguagem de programação que possibilita a criação de *scripts* para 3Ds Max.

[11] Nomeclatura da UE para modelos 3D que não possuem hierarquia de *bones* em sua estrutura.

visual encontrada que funciona de forma semelhante ao *material editor* possuindo um sistema de *nodes* próprios ao mesmo.

Figura 43 Interface sistema de blueprints.

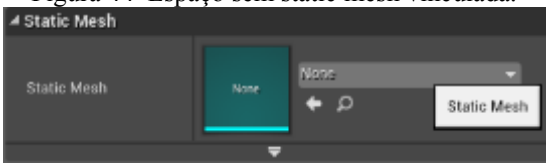


Fonte: Autora.

A adaptação dos elementos de cenário que eram animáveis dentro do 3Ds Max para a UE acabou sendo descartada, pois os objetos em questão não seriam animados nas cenas que seriam montadas e trabalhar com o sistema de *blueprints* não fazia parte dos objetivos do projeto em questão.

Importados os elementos, então, eles foram posicionados na cena. Dependendo da complexidade da cena o posicionamento pode ser feito de forma manual. Como a intenção do projeto era replicar o cenário da forma como havia sido organizado no 3Ds Max, foram copiadas as posições dos elementos no 3Ds Max e "coladas" dentro do projeto na UE, com o auxílio de *maxscripts* para permitir a cópia exata da posição no 3Ds Max e sua conversão para UE. Esse método cria um espaço vazio onde a posição foi "colada" permitindo que seja designada uma *static mesh* ao mesmo.

Figura 44 Espaço sem static mesh vinculada.



Fonte: Autora.

4.2 ADAPTAÇÃO DOS MATERIAIS DO VRAY PARA A UE

A série "Dias Velho" estava sendo produzida para ser renderizada em VRay, então, todos os personagens e cenários estavam configurados em materiais próprios do VRay acabariam se perdendo nessa adaptação,

foi-se proposto que se tentasse recriar algo próximo desses materiais dentro da UE.

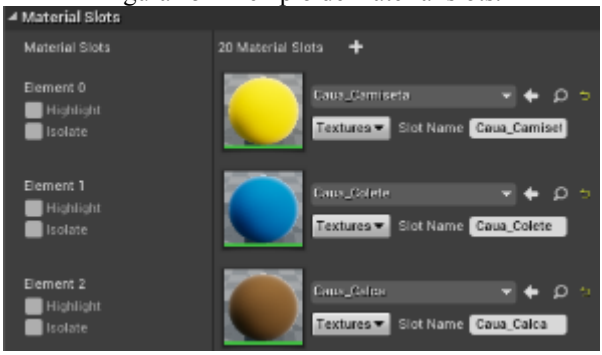
Vale comentar que, atualmente, já existem algumas ferramentas como o VRay para Unreal e Unreal Datasmith que fazem essa conversão de materiais automaticamente, porém ainda se encontram em fase de testes e ainda precisam de um maior refinamento, além disso, após a fase *beta*, ambas serão disponibilizadas para venda, logo, poderão acrescentar custos ao projeto de quem desejar usá-las.

A conversão dos materiais foi feita manualmente levando-se em conta a característica estética principal do projeto de ser uma série com uma aparência mais *cartoon*, logo, o estilo de render seria algo mais estilizado e não foto-realista.

4.2.1 Personagens

Apesar dos materiais se perderem na importação para dentro da UE, a configuração das *material ids*[12] não é perdida e são importadas para dentro do programa como *material slots* que podem ser visualizadas abrindo o menu da *skeletal mesh*. Caso essa configuração não houvesse sido feita previamente todo o personagem seria importado com apenas um *material slot* e a divisão de que material corresponde a qual região do modelo deveria ser feita no 3Ds Max para depois reimportar o modelo para UE.

Figura 45 Exemplo de material slots.



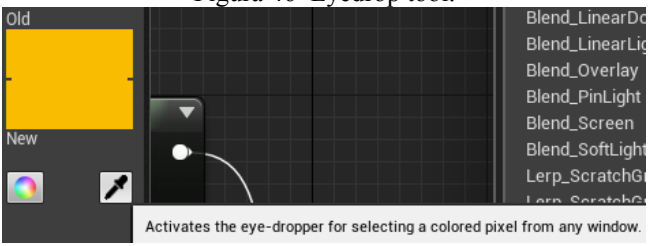
Fonte: Autora.

Para começar a configurar os materiais, primeiro adicionou-se as texturas, que já estavam prontas, e cores correspondentes a cada um dos materiais. O *node Costant3Vector* (que é um *node* do tipo constante onde

[12] Configuração do 3Ds Max que permite a atribuição de materiais a determinados conjuntos de polígonos em um modelo.

três valores podem ser atribuídos ao objeto como no sistema RGB) utilizado para configurar as cores possui um *eyedrop tool* em seu *color picker* que pôde ser utilizado para pegar as cores dos materiais do VRay e transferi-las para a UE.

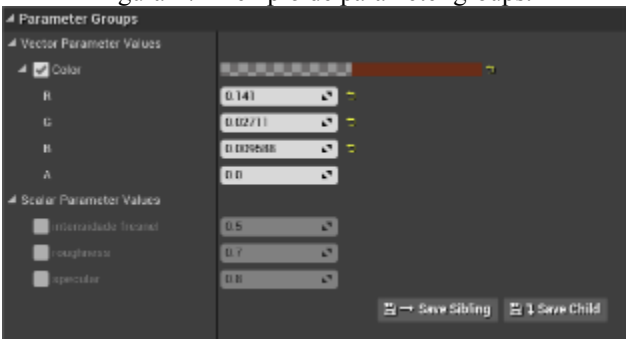
Figura 46 Eyedrop tool.



Fonte: Autora.

Também se avaliou quais materiais poderiam ser configurados como *material instances*, isto é, materiais que podem ser repetidos com a alteração de alguns parâmetros, estando todos vinculados a um mesmo material base, alguns parâmetros que podem ser configurados para serem alterados nas *instances* são, por exemplo: cores, texturas, valores de *specular* e *roughness*, entre outros.

Figura 47 Exemplo de parameter groups.



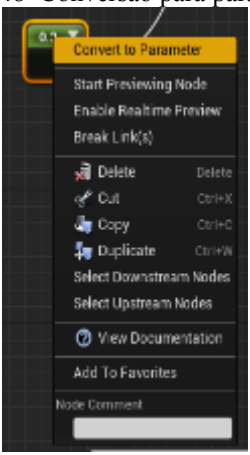
Fonte: Autora.

Esses materiais são úteis quando a cena possui muitos materiais que possuem características semelhantes exceto por algumas pequenas diferenças como a cor, por exemplo. Transformar o material em uma *instance* de outro faz com que a *engine* precise calcular menos informações quando o material for compilado na cena, já que a única informação que o programa precisará calcular será a cor diferente.

Para configurar as *material instances* transformou-se os *nodes* dos atributos que desejavam ser modificados nas *instances* para *parameters*

no caso de nodes do tipo constante (como o anteriormente mencionado *Constant3Vecto*r) e *texture objects* no caso de *texture samples*.

Figura 48 Conversão para parâmetro.



Fonte: Autora.

Após as cores e texturas serem devidamente configuradas, avaliou-se as outras principais características presentes nos materiais do VRay de forma que a adaptação dos materiais fosse fiel a estética do projeto. Percebeu-se então que os materiais possuíam duas características:

- Mapas de *falloff* que acrescentam uma sutil iluminação nas extremidades dos modelos (principalmente personagens), ajudando a ressaltá-los na cena.
- Mapas de gradiente configurados para dar um efeito de *cel shading* nos materiais.

Foi pesquisado qual *node* no editor de materiais da UE poderia dar um efeito semelhante aos mapas de *falloff* do VRay e conclui-se que seriam um dos *nodes* de fresnel disponíveis na UE, chamado de *Fresnel Utility*. A documentação da UE define fresnel como "termo usado para descrever como a luz que você vê reflete a diferentes intensidades baseada no ângulo em que você está vendo-a"(UNREAL ENGINE, 2015).

No caso do projeto o efeito foi utilizado ajudar a criar a iluminação pretendida sem ter a necessidade de adicionar outras luzes na cena, que poderiam aumentar o tempo de rendering.

Para tentar reproduzir a característica do segundo ponto, pesquisou-se sobre maneiras de reproduzir um sombreamento estilo *cel shading* dentro da UE. Foram encontradas duas possíveis formas com a primeira sendo através de um material que divide o sombreamento em bandas, semelhante ao que o mapa de gradiente fez no material do VRay,

e a segunda sendo através da configuração de um *Post Process Material* aplicado a um *Post Process Volume* que cobriria a cena e daria a mesma um aspecto *cel shading* (o *Post Process Volume* será explicado em mais detalhes no tópico referente a pós-produção).

As duas formas foram testadas e concluiu-se que: o material especial atingiu algo próximo ao efeito desejado. O processo de configuração para um material com esse tipo de sombreamento seria diferente em um objeto que apenas possui cor base de um objeto com textura, pois alguns do nodes utilizados para alcançar efeito são incompatíveis com os nodes de textura da UE, tornando sua configuração complexa.

Figura 49 Teste material estilo cel shading.

Fonte: Autora.

A outra forma seria um *Post Process Material* aplicado a um *Post Process Volume* configurado para afetar a cena como um todo, tornando-o mais fácil de configurar. A desvantagem foi a dificuldade de controlar o efeito de forma a obter a estética desejada, já que o *Post Process Material* testado possuía uma configuração que gerava um efeito de posterização. A separação do sombreamento em bandas acontecia porém não era próximo da qualidade atingida pelo VRay.

Figura 50 Teste cel shading via post process.

Fonte: Autora.

Nenhuma das técnicas acabou sendo empregada por não atingirem os resultados esperados. Outras possibilidades de solução para o problema entrariam na área de configuração de *custom shaders*, algo mais voltado para área da programação e além da proposta do projeto.

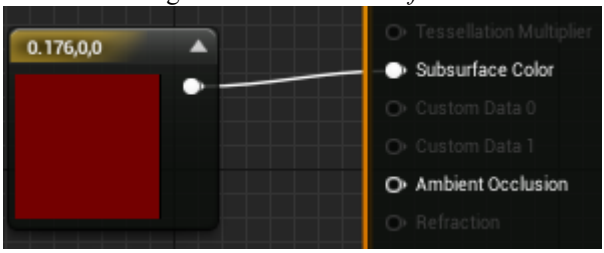
Por fim, avaliou-se que configurar os materiais com altos valores de *roughness* e baixos de *specular* davam um aspecto mais *cartoon* aos materiais, encaixando-os na estética do projeto, mesmo com sombreamento divido em bandas não ter sido possível.

Na maioria dos materiais apenas foram apenas adicionadas as cores ou texturas e a configuração de fresnel (além das configurações de *roughness* e *specular*), porém alguns materiais específicos foram configurados em algumas partes como: pele de ambos os personagem, lente do óculos do personagem Cauã e a renda da camisa da personagem Leca.

Na pele utilizou-se um efeito de *subsurface scattering*[13], quando o *blend mode* de *subsurface* é selecionado na UE ele abre no material a opção de adicionar uma *subsurface color*, isto é, a cor que será vista nas extremidades de onde a luz passar. A cor escolhida foi um vermelho escuro para simular a cor do sangue humano.

[13] Fenômeno de dispersão da luz quando a mesma atravessa um objeto translúcido ou semi-translúcido (UNREAL ENGINE, 2015).

Figura 51 Cor do *subsurface*.

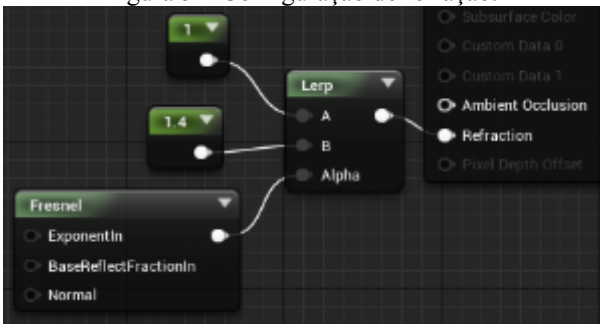


Fonte: Autora.

Entretanto, a intenção não era deixar a personagem com uma pele foto realista, suas características *cartoon* se mantiveram graças à textura que havia sido pintada previamente e foi aplicada no material da personagem, o efeito *subsurface* apenas adicionou um refinamento ao material para deixá-lo mais visualmente interessante.

Para a lente dos óculos criou-se um material que simula vidro, para isso usou-se o *blend mode* de *translucent* da UE e na aba *translucency* configurou-se o *lighting mode* como *surface translucency volume* (essa configuração é importante pois sem ela o material não disponibiliza todas as opções necessárias para criar o vidro). Então, utilizando novamente o *node* de fresnel configurou-se a refração do vidro.

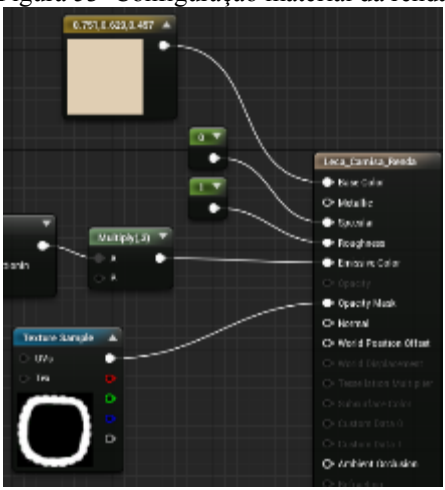
Figura 52 Configuração de refração.



Fonte. Autora.

Para a renda da camisa da personagem Leca, foi aplicada uma máscara de opacidade, isso pôde ser facilmente reproduzido na UE com um material do tipo *masked* onde a máscara de alpha pintada para ser utilizada do material do VRay pode ser reaproveitada.

Figura 53 Configuração material da renda



Fonte:Autora.

Nos materiais onde a suavização feita pelos *smoothing groups* não foi suficente e foi necessária uma suavização a mais foi adicionado *tessellation* ao material. O processo de *tessellation* é um método de subdivisão de malha que pode ser aplicado dentro da *engine,* onde os triângulos do modelo são divididos em triângulos menores, na UE existem duas opções de *tessellation*: *flat* e *PN triangles*.

A opção utilizada no projeto foi a *PN triangles* que é a opção que efetivamente suaviza a malha, para que ela funcione é necessário que o objeto possua *smoothing groups*. Vale comentar que para o *tessellation* funcionar corretamente na UE é necessário que o computador possua um hardware com suporte para DirectX 11 (UNREAL ENGINE, 2015).

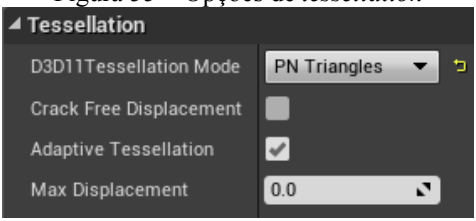
Figura 54 Bota da personagem Leca antes e depois do tessellation

Fonte:Autora.

É possível configurar o *tessellation* do material para ser adaptável através da opção *adaptive tessellation*, isto é, o efeito se adapta a distância que o objeto está da câmera, se está perto há mais subdivisões para ajudar

a suavizar, se está longe há menos subdivisões. Isso existe, pois, como tudo que aumenta o número de triângulos do modelo, o efeito afetará o tempo em que o computador levará para calcular o render.

Figura 55 – Opções de *tessellation*



Fonte:Autora.

Com o *adaptive tessellation,* então, o efeito apenas atinge sua capacidade máxima em situações onde realmente exigem um maior nível de detalhamento como quando a câmera está próxima, exigindo menos do computador quando a câmera está longe.

Figura 56 Render final dos personagens

Fonte: Autora.

4.2.2 Cenários

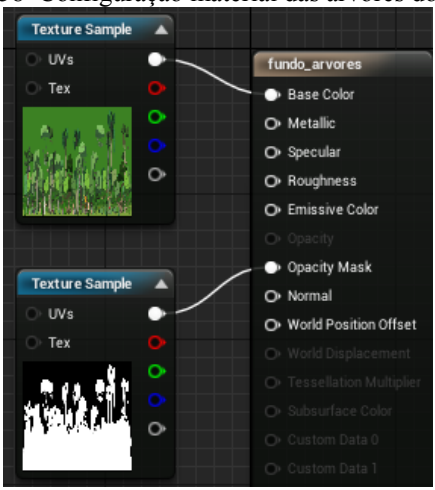
O material do cenário passou por um processo semelhante ao dos personagens, começando por adicionar as cores e texturas nos objetos e configurando as *material instances*. Um exemplo de onde se utilizou *material instances* foram os troncos e as folhas das árvores. Ao invés de criar um novo material para cada uma delas apenas criou-se um material principal para os troncos e um principal para as folhas apenas trocando-se as texturas.

Figura 57 Materiais dos troncos.

Fonte: Autora.

As árvores do fundo do cenário eram pinturas 2D onde se aplicou uma máscara de opacidade para que tivessem fundo transparente, como foi feito previamente no material da renda da camisa da Leca.

Figura 58 Configuração material das árvores do fundo.

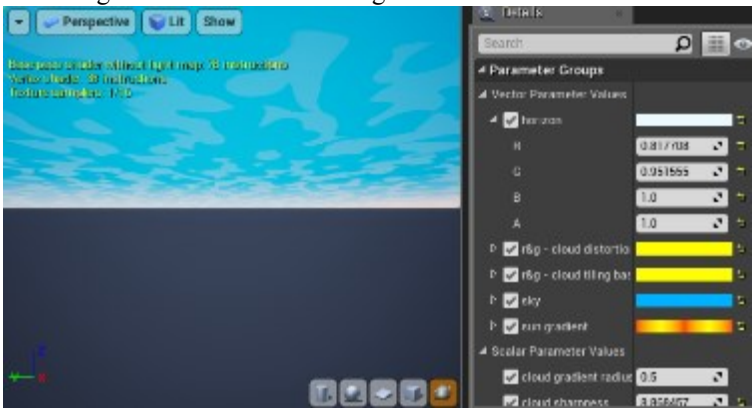


Fonte: Autora.

O céu no cenário original era uma textura pintada aplicada a um plano que circundava o cenário, uma das propostas na adaptação dos materiais era reproduzir da forma mais fiel possível os materiais do VRay, porém também havia uma abertura para melhorar alguns materiais desde que não fugisse da estética da série.

Para o céu, então, criou-se um material que simula nuvens em um estilo de render mais próximo ao *cartoon*, porém elas são animadas e seu tamanho e velocidade podiam ser configurados. Esse material foi baseado no material do céu utilizado no projeto de demonstração *Stylized Rendering* da UE, um dos vários projetos disponibilizados gratuitamente pela equipe da Epic Games para fins de estudo e aplicação em projetos próprios.

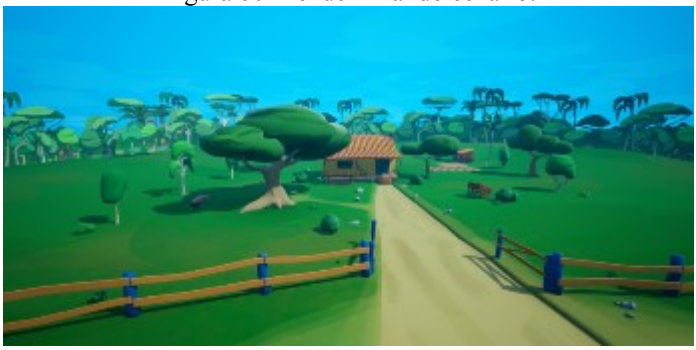
Figura 59 Parâmetros configuráveis do material do céu..



Fonte: Autora.

O material foi configurado com diversos *nodes* transformados para parâmetro e então uma *instance* desse material foi criada. O único material efetivamente aplicado à cena foi à cena final foi essa *instance,* que foi aplicada a um plano circular curvado que cobre o cenário todo, escondendo o céu pré-configurado que existe na própria *engine*. Nesse caso, o material original serviu apenas como base para criar os parâmetros que irão configurar o tamanho das nuvens, distância entre elas, velocidade, etc. Isso mostra outra possibilidade de utilização de *material instances* em uma cena.

Figura 60 Render final do cenário.



Fonte: Autora.

Figura 61 Render final: detalhes casa.

Fonte: Autora.

Figura 62 Render final: fundos da casa.

Fonte: Autora.

4.3 EXPORTAÇÃO DAS ANIMAÇÕES EM FBX E ALEMBIC

Para as animações decidiu-se utilizar um *workflow* híbrido entre fbx e alembic inspirado na técnica utilizada pelos artistas da Epic Games para o trailer do jogo *Fortnite*(2017) onde as animações em fbx foram utilizadas para o corpo e em alembic para o rosto

O alembic é um formato de arquivo desenvolvido em conjunto pelos estúdios *ILM (Industrial Light & Magic)* e *Sony Pictures Imageworks* para ajudar na transferência de animações mais complexas como simulação de tecidos e pele entre programas através de um sistema que funde a animação à geometria do modelo.

A vantagem do alembic é a possibilidade de importar animações mais refinadas, já que não existe um limite de influências que podem ser exercidas em cada vértice ao contrário do formato fbx (POHL et al., 2018), entretanto, o formato alembic é mais pesado e consome mais

recursos da máquina por isso é recomendado que ele seja apenas utilizado para animações difíceis de serem exportadas via fbx.

A UE possui suporte ao formato alembic desde a versão 4.13 e permite que arquivos do formato sejam importados como *Static Mesh*, *Skeletal Mesh* e *Geometry Cache* (ainda em fase experimental).

4.3.1 Exportação e importação de animações via fbx

A exportação da animação dos corpos foi feita via 3Ds Max, para isso, selecionou-se apenas o CAT dos personagens já que era onde se encontravam os *keyframes* das animações. Os objetos 3D que estavam vinculados à hierarquia do CAT foram retirados da seleção.

Normalmente seria possível manter esses objetos na hierarquia já que é possível importá-los para dentro da UE através do menu importador de animações, entretanto, verificou-se que o modelo do personagem Cauã que havia sido usado nas cenas de onde as animações foram retiradas ainda possuía os problemas de escala que foram corrigidos no personagem original.

Quando as animações eram exportadas com esses objetos de escala errada, o arquivo apresentava erros de *bone transform* já que a escala do objeto na hierarquia estava diferente do objeto da hierarquia na *skeletal mesh* que havia sido importada para UE.

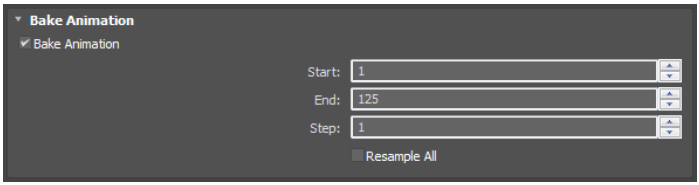
Ao invés de corrigir todos os problemas de escala de todas as cenas para ficar igual ao Cauã importado, concluiu-se que seria mais eficiente apenas retirar esses objetos da hierarquia de *bones* para que a animação importasse corretamente.

Figura 63 Exemplo do erro de bone transform.

Fonte: Autora.

Com a seleção feita, a animação foi exportada através do *export selected* do 3Ds Max. O mesmo menu que foi visto nas outras exportações abrirá novamente, porém o mais importante dessa vez foi selecionar a aba *animation* e, dentro dela, *bake animation*, então foi selecionado o segmento de frames de animação que seria capturado para exportação.

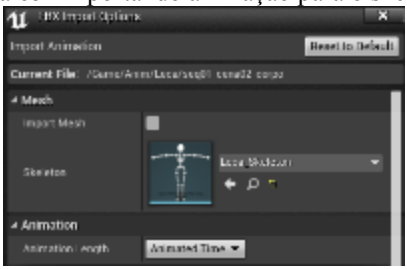
Figura 64 Opções de bake animation.



Fonte: Autora.

O arquivo é importado para a UE como *skeletal mesh*, porém, como apenas desejava-se a animação do personagem a opção *import mesh* foi deselecionada. O esqueleto selecionado foi o *skeleton asset* criado quando cada personagem foi importado para dentro da UE. As animações correspondentes à personagem Leca foram vinculadas ao *skeleton asset* dela e as animações do Cauã ao *skeleton asset* dele, então, quando o arquivo da animação é aberto dentro da UE é possível já vê-las aplicadas ao modelos do personagens.

Figura 65 Importando animação para o skeleton.



Fonte: Autora.

Figura 66 Animação aplicada a skeletal mesh da Leca.

Fonte: Autora.

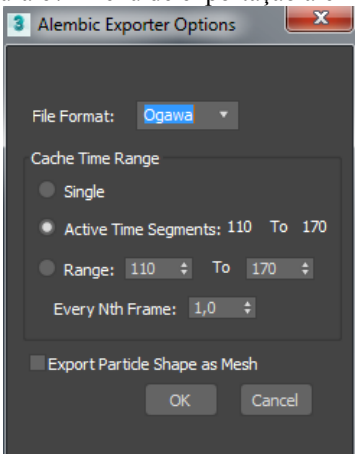
4.3.2 Exportação e importação de animações via alembic

Como mencionado previamente o alembic foi apenas utilizado para a animação das expressões faciais dos rostos dos personagens, a exportação foi feita do 3Ds Max.

O rosto do personagem o qual a animação seria exportada foi selecionado. Nesse caso, o *turbosmooth* foi mantido e a malha foi exportada já suavizada.

O rosto também foi exportado através do *export selected*, sendo selecionado o formato alembic ao invés de fbx. O programa abre um menu no qual é possível selecionar o segmento de animação, na figura abaixo é possível ver as configurações utilizadas na exportação de uma das animações utilizadas no projeto.

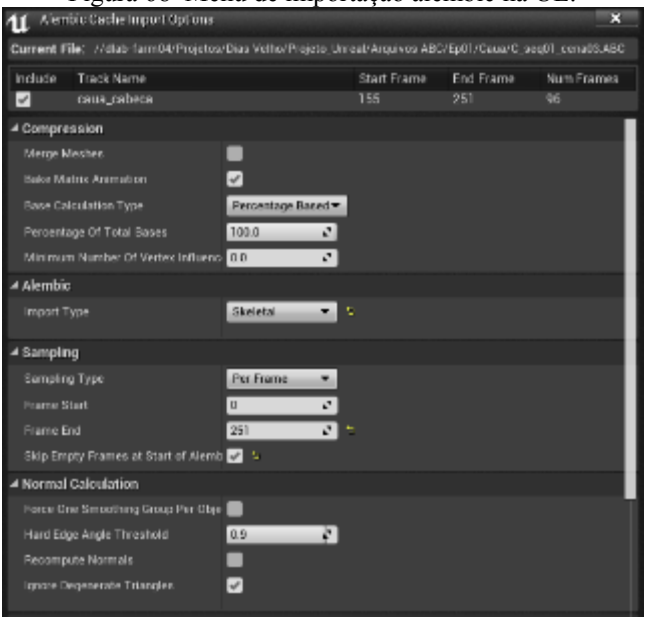
Figura 67 Menu de exportação alembic.



Fonte: Autora.

Ao importar para a UE o arquivo já é identificado como alembic e o menu de opções é aberto. As animações do projeto foram importadas em *skeletal mesh* devido ao formato possuir uma melhor integração com a ferramenta Sequencer do que o *geometry cache*.

Figura 68 Menu de importação alembic na UE.



Fonte: Autora.

4.4 MONTANDO AS CENAS DENTRO DO SEQUENCER

Para começar a montar as cenas foi criada uma *master sequence*, uma sequência principal contendo todas as cenas que serão trabalhadas. Ao contrário da *level sequence* que necessita que todas as câmeras e cenas sejam inseridas na sequência, na *master sequence* é possível automatizar esses processos, tornando-a ideal de se trabalhar quando já se possui um *storyboard* ou *animatic*, como era o caso.

As cenas escolhidas para o projeto foram as quatro primeiras cenas da primeira sequência do episódio piloto da série, onde aparecem os personagens Leca e Cauã brincando de piratas no quintal da casa do Dias Velho. Assim, a *master sequence* foi configurada para possuir quatro cenas e o tempo de duração pré-configurado foi retirado do *animatic* da sequência, fazendo com que ela já viesse com quatro cenas divididas e um *cine camera actor* atrelada a cada uma delas.

Antes de se adicionar os personagens e suas respectivas animações, a interface da UE foi configurada para facilitar o trabalho no Sequencer. Para isso, o layout da *viewport* foi dividido para ter duas vistas, com a segunda visa sendo transformada em uma *cinematic viewport*.

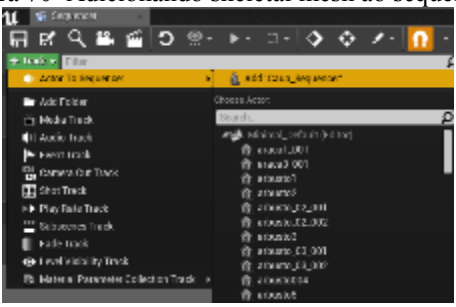
Figura 69 Cena vista na cinematic viewport.

Fonte: Autora.

A *cinematic viewport* facilita na hora de se fazer os enquadramentos de cena já que nos permite ter a vista da câmera, através da opção de *lock viewport to shot*, e também possui a possibilidade de incluir guias visuais.

No Sequencer foram incluídos os personagens animados da sequência. O processo se deu adicionando a *skeletal mesh* dos personagens (em sua versão sem cabeça previamente importada) ao cenário e, com o personagem selecionado, adicioná-lo a sequência através da opção *add actor to sequencer*. É possível também adicioná-los sem que estejam selecionados apenas clicando no *add actor to sequencer* e buscando o nome do *actor* na lista. É importante certificar-se de que a sequência encontra-se ativa dentro do Sequencer caso contrário não será possível adicionar o personagem à cena.

Figura 70 Adicionando skeletal mesh ao sequencer.



Fonte: Autora.

Os personagens foram adicionados dentro de cada uma das quatro cenas, entretanto, não é necessário adicionar a *skeletal mesh* várias vezes

ao cenário uma vez que várias cenas podem compartilhar a mesma *skeletal mesh*, sendo apenas necessário que elas sejam adicionadas à cena através do *add actor to sequencer* para permitir que as animações sejam inseridas depois.

Quando *skeletal meshes* são adicionadas como *actors* ao Sequencer é possível ver a aba *animation*, que permite a inclusão de animações. Porém, antes a posição dos personagens foi zerada porque quando as animações são adicionadas o *pivot* do personagem muda para o *pivot* da animação que se encontrará no ponto zero da cena na UE, semelhante ao que acontece na importação de objetos quando o mesmo não tem sua position zerada antes da exportação do 3Ds Max.

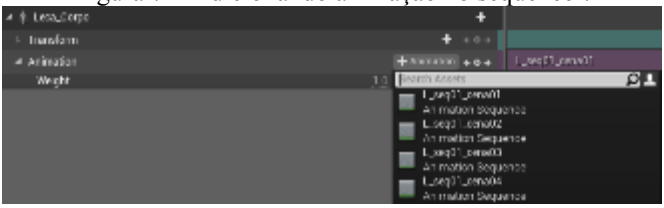
Figura 71 Corrigindo a posição.

Fonte: Autora.

Nos objetos, o *pivot* se deslocar para ponto zero da UE era indesejável, pois dificultaria o trabalho do artista caso ele deseje fazer alguma alteração futuramente no posicionamento do objeto. No caso das animações, o *pivot* deslocado é útil para posicionar o personagem exatamente no mesmo local onde ele foi configurado quando a animação foi feita no 3Ds Max.

Com a posição dos personagens zeradas, foi adicionada a animação através da do botão + *animation* e selecionou-se a animação correspondente à cena que estava sendo trabalhada. Os arquivos de animação já haviam sido previamente nomeados com o nome da cena a qual correspondia para facilitar esse processo.

Figura 72 Adicionando animação no sequencer.



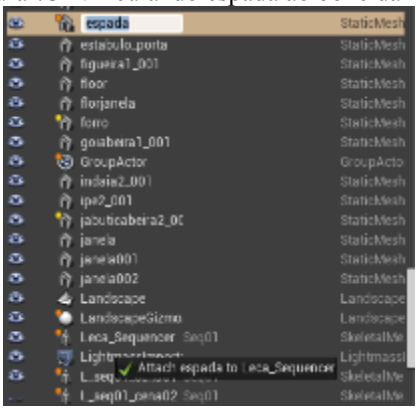
Fonte: Autora.

Com os rostos foi feito o mesmo processo, entretanto, o rosto de cada cena é uma *skeletal mesh* separada, já que não foi possível encontrar uma forma de um único rosto compartilhar todas as animações visto que

o formato alembic não exporta a hierarquia de *bones* que seria necessária para tentar executar um processo de *retarget*[14] nas animações.

A personagem Leca segurava uma espada nas cenas escolhidas, a mão já estava posicionada corretamente na animação sendo apenas necessário fazer com que o objeto "grudasse" na mão de forma a seguir a animação. A espada havia sido importada como *static mesh* e foi executado um *attach to bone* a um dos *bones* da mão que se seguraria a espada, para então posicionar a mesma na mão da personagem.

Figura 73 Vinculando espada ao bone da mão.



Fonte: Autora.

É essencial que a *static mesh* esteja marcada como *movable* nas opções de *mobility* caso contrário esse processo não funcionará já que um *actor* dinâmico não pode ser unido a um *actor* estático (em relação à mobilidade do *actor*, não ao tipo de *mesh*).

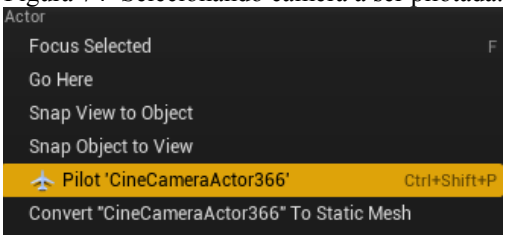
Com os personagens e rostos inclusos dentro das cenas no Sequencer, junto com suas respectivas animações posicionadas no tempo correto, deu-se início ao processo de configurar os enquadramentos de câmera. Como mencionado anteriormente, quando se cria uma *master sequence* já possuímos um *cine camera actor* incluso em cada cena então não há necessidade de incluir uma câmera a mesma, caso a cena ainda não possuísse uma câmera ela seria incluída em um processo semelhante ao feito para as *skeletal meshes*.

Ao clicar na câmera é possível ver na aba *details* as configurações de tipo lente, foco, entre outras. Na animação foi utilizado formato 16:9

[14] Ferramenta da UE que permite configurar que uma mesma animação seja compartilhada por dois personagens com o mesmo *skeleton*, mas diferentes proporções, ou por dois *skeletons* diferentes.

Digital Film. O posicionamento da câmera pode ser feito através das ferramentas de posicionamento, como também através do *pilot camera*. Quando o *pilot camera* está ativado a forma a qual nos posicionamos na *viewport* influencia na posição da câmera.

Figura 74 Selecionando câmera a ser pilotada.



Fonte: Autora.

A câmera foi pilotada através da *viewport* comum e o resultado do enquadramento era visto através da *cinematic viewport* com a opção *lock camera to shot* ativada para cena, caso a cena seja trocada é importante selecionar a mesma opção na outra cena, já que essa alteração não é feita automaticamente.

Para ver o resultado de todas as cenas da sequência apenas volta-se para a *master sequence* e seleciona-se a mesma opção na aba *camera cuts*, permitindo a visualização da sequência completa na *cinematic viewport*. Após a configuração dos enquadramentos foram feitos os últimos ajustes de foco nas lentes.

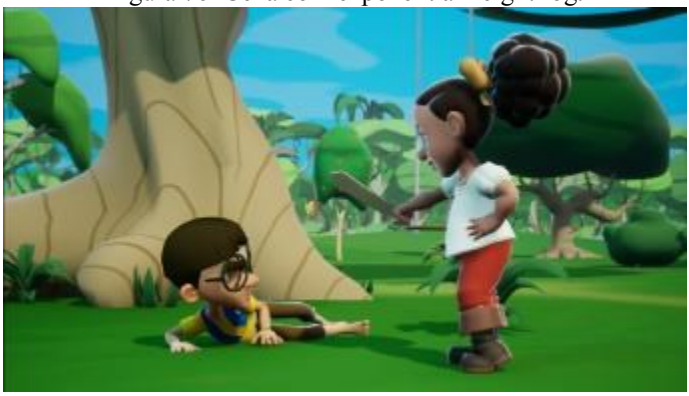
Figura 75 Trabalhando com as duas viewports.

Fonte: Autora.

A iluminação das cenas se dá por duas luzes, uma *sky light* e uma *directional light* funcionando como luz do sol. Como o fresnel configurado nos materiais já havia alcançado o efeito de iluminação que ajuda os personagens a destacarem-se da cena, não houve a necessidade de adicionar uma outra luz a cena de forma a atingi-lo.

Para aumentar a sensação de profundidade à cena, foi adicionado um *actor* chamado *exponetial height fog*, que pode ser encontrado na aba de *visual effects* da UE. Na sequência o efeito serviu para deixar as regiões mais afastadas, isto é, a região onde se encontra o condensado de árvores, encoberta por uma leve "neblina", dando a impressão que as árvores estão ainda mais distantes.

Figura 76 Cena com exponential height fog.



Fonte: Autora.

4.5 PÓS-PRODUÇÃO

Existem ferramentas que possibilitam que a pós-produção ou, ao menos, parte dela (dependendo da complexidade do projeto), sejam feitas dentro da própria *engine*. Isto faz com que não haja a necessidade de que a animação seja renderizada para então ser enviada a um programa de edição de vídeo para ajustes e refinamentos de imagem.

Esses ajustes podem ser feitos dentro da UE de duas formas, ou ajustando as configurações de cada câmera (as ferramentas de *color grading* são encontradas no *cine camera actor*) individualmente ou através de um *Post Process Volume* (PPV). A principal diferença sendo que as câmeras irão afetar cada *shot* individualmente enquanto o PPV pode ser configurado para afetar todo o *level*[15] como um todo.

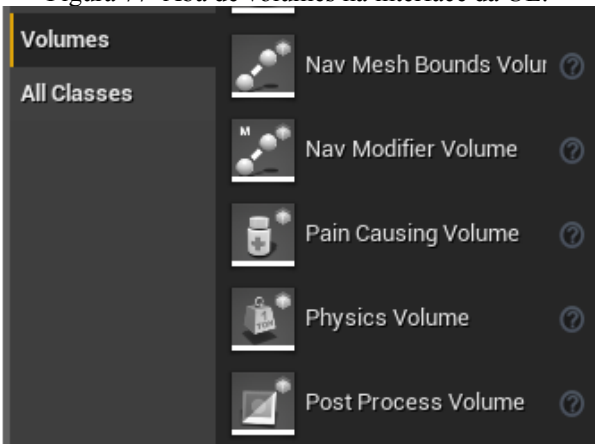
O principal problema a ser resolvido nesse ponto era o de que a imagem parecia mais estourada nas primeiras três cenas e escura na cena número quatro. A solução foi regular a exposição e diminuir um pouco do brilho e saturação nas primeiras cenas. Na cena que estava escura,

[15] *Asset* onde se encontram o cenário, os personagens e os elementos com os quais estes podem vir a interagir. Um mesmo projeto pode possuir diversos *levels* vinculados a ele, cada um com características próprias.

elevaram-se um pouco os tons médios e os brancos através da ferramenta de *color grading*, ajustando a saturação e brilho de forma a não deixar a cena com aparência “lavada”.

Para que isso fosse possível, utilizou-se um PPV. O PPV é um dos diversos tipos de *volumes* encontrados na UE. *Volumes* são *actors* tridimensionais que podem controlar determinados comportamentos dentro de um *level* (UNREAL ENGINE, 2015). No caso do PPV ele pode controlar aspectos de como a imagem é renderizada como: exposição, cores, iluminação global, entre várias outras opções que ele disponibiliza, sendo também possível vincular um *Post Process Material* a ele.

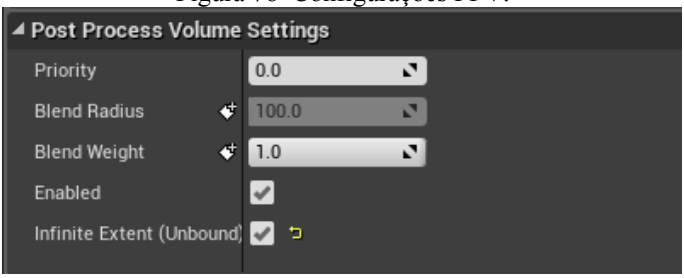
Figura 77 Aba de volumes na interface da UE.



Fonte: Autora.

Por ser um *volume*, é possível configurar um limite em seu tamanho, é possível também adicionar mais de um PPV a cena dependendo das pretensões artísticas do projeto. Entretanto, como mencionado anteriormente, a opção por esse *volume* foi feita devido a possibilidade de afetar o *level* inteiro, então se selecionou a opção *infinite extent (unbound)* nas configurações do *volume*.

Figura 78 Configurações PPV.

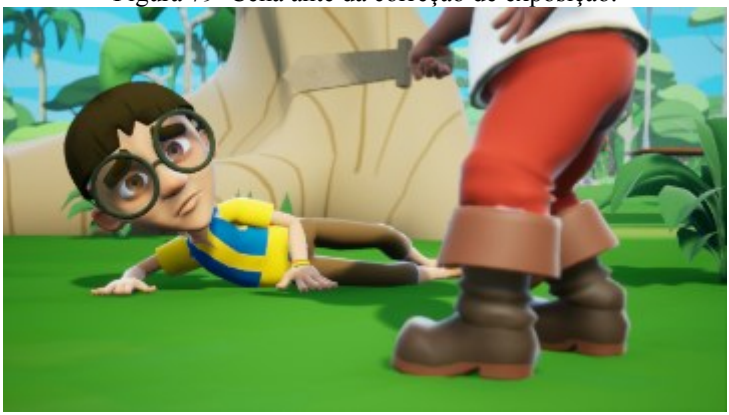


Fonte: Autora.

É possível adicionar o PPV ao Sequencer, diversas configurações dele são animáveis através de *keyframes*. As configurações de *color grading* foram animadas de forma que se mantivessem as mesmas para as três primeiras cenas e mudassem quando entrassem na quarta, todas as outras alterações feitas no PPV foram mantidas iguais para todas as cenas.

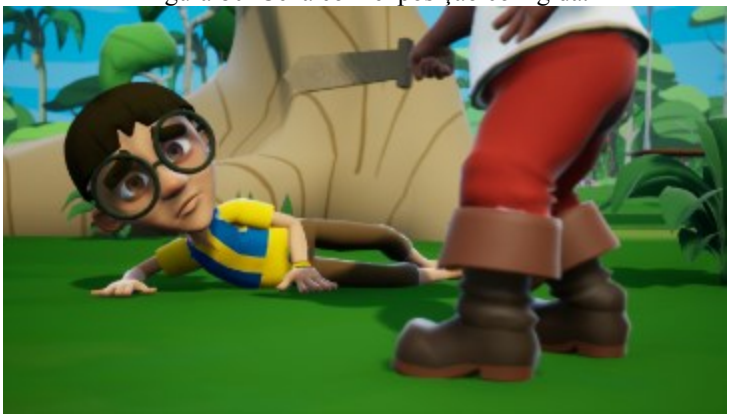
As opções de exposição da UE permitem que sejam configuradas uma exposição mínima e uma máxima, optou-se por manter a mínima e a máxima no mesmo número de forma a não ter grandes diferenças de exposição entre uma cena e outra.

Figura 79 Cena ante da correção de exposição.



Fonte: Autora.

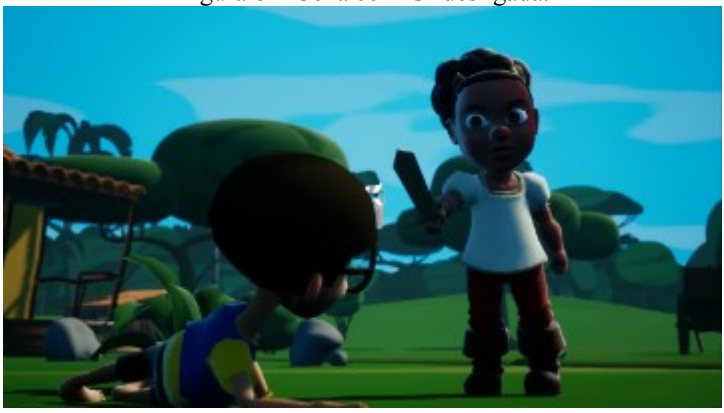
Figura 80 Cena com exposição corrigida.



Fonte: Autora.

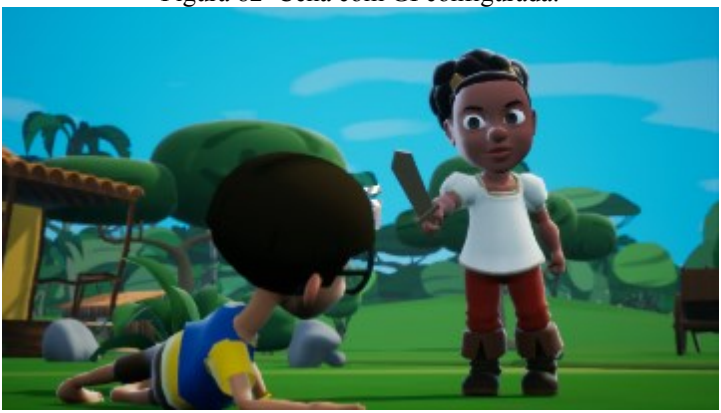
As configurações de Iluminação Global do PPV permitem que sejam configuradas a intensidade e cor da luz indireta, a UE já vem com a intensidade de 1 como padrão, sendo que 0 tira toda a GI da cena. A intensidade foi aumentada um pouco para dar mais vida à cena e a cor mantida como branco padrão.

Figura 81 Cena com GI desligada.



Fonte: Autora.

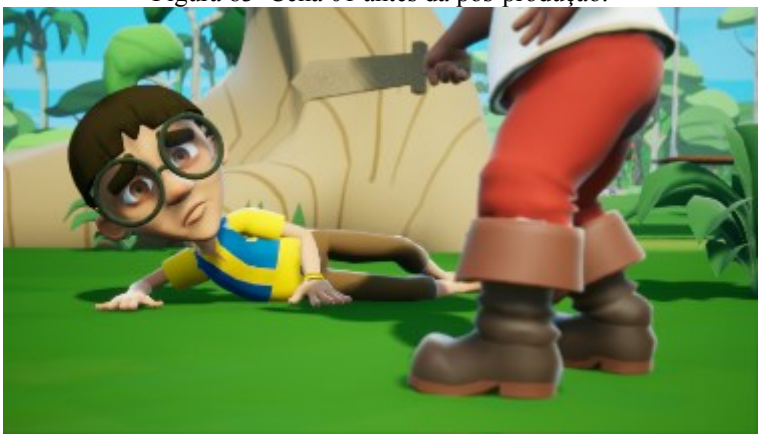
Figura 82 Cena com GI configurada.



Fonte: Autora.

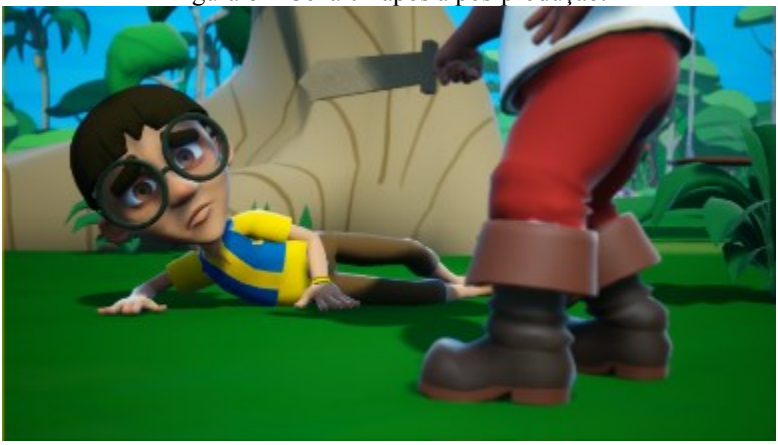
Para a correção de cor seguiu-se a recomendação de *workflow* apresentado pela documentação da UE, primeiramente configurando o tom geral da cena através do *film tonemapper* e então fazendo o refinamento de cor individual das cenas através da ferramenta de *color grading*, que foi animada no Sequencer para que as três primeiras cenas mantivessem as mesmas configurações de cor, apenas mudando quando chegasse à quarta cena.

Figura 83 Cena 01 antes da pós-produção.



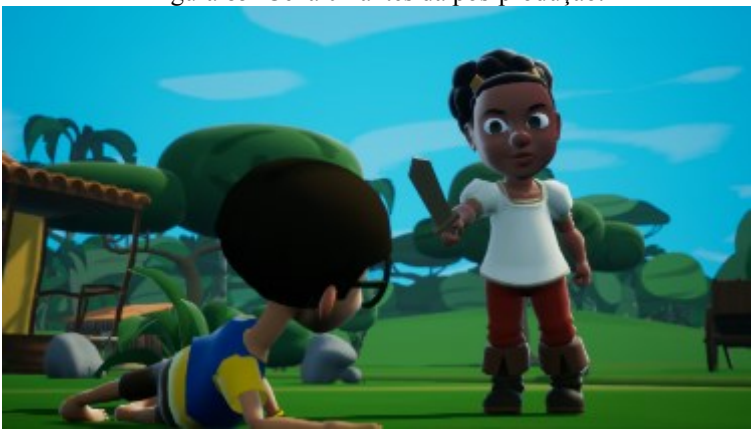
Fonte: Autora.

Figura 84 Cena 01 após a pós-produção.



Fonte: Autora.

Figura 85 Cena 04 antes da pós-produção.



Fonte: Autora

Figura 86 Cena 04 depois da pós-produção.

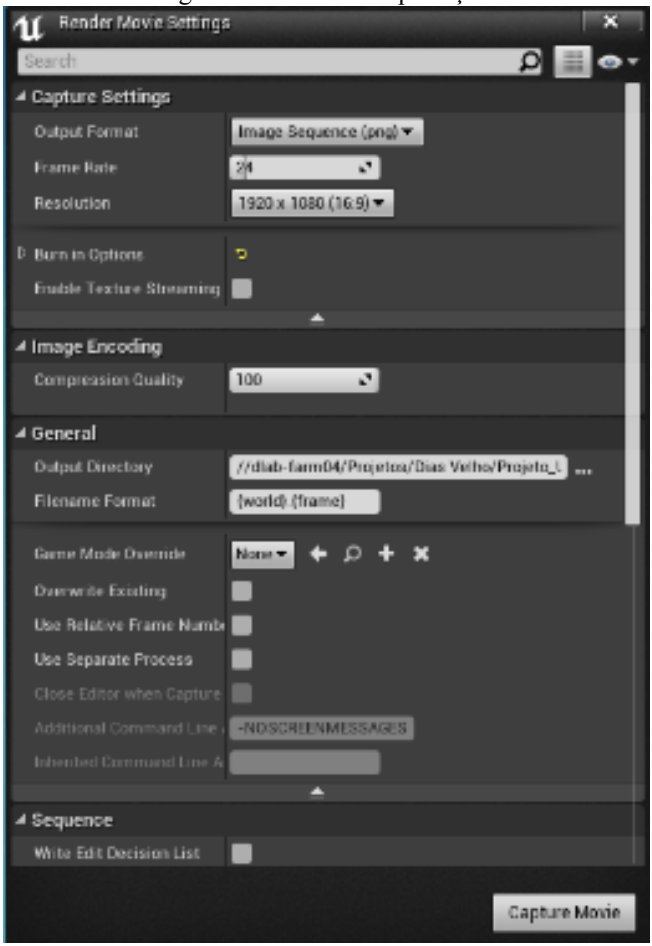
Fonte: Autora.

Após as cenas totalmente montadas, a dublagem foi adicionada. É possível adicionar áudio dentro do Sequencer importando o áudio para o projeto (importante notar que o único formato aceito pela *engine* é wav 16 bits) e adicionando o mesmo a sequência através de um *audio track*.

Para concluir, adicionou-se um efeito de *fade* através de um *fade track*, o efeito criado é simples, análogo ao *fade to black* do Adobe Premire, e é o único efeito de transição atualmente presente no Sequencer. Então, caso a animação necessitasse de efeitos mais refinados o recomendado seria exportar a animação para um programa de edição de vídeo.

É possível exportar a animação como sequência de imagens ou vídeo através da ferramenta de renderização do Sequencer.

Figura 87 Menu de exportação.



Fonte: Autora.

5 CONCLUSÃO

No geral, é possível considerar o projeto bem sucedido, pois se conseguiu incluir um número maior de personagens e cenas do que havia sido estipulado nas fases iniciais. Concluiu-se que, uma vez que os modelos e materiais estejam prontos, o processo até a saída da cena final é mais rápido que em um método que faz uso de rendering *off-line*.

Também se considerou o projeto bem sucedido em manter a estética da série e uma qualidade de render comparável a que estava sendo obtida via VRay. O único ponto que não foi possível resolver foi à questão do efeito *cel shading*, de forma que se aproximasse ao que estava sendo usado na série. Para isso, talvez seria necessário fazer um estudo mais aprofundado de como se deva configurar *post process materials*.

Avaliou-se que, com os *assets* do projeto em questão, o sistema de *tessellation* da UE nem sempre se comportou da forma desejada, causando alguns problemas como um erro na camisa da personagem Leca. Logo, neste caso seria mais interessante fundir o *turbosmooth* à malha e exportar o modelo suavizado, o que não foi testado, pois envolveria o trabalho de refazer os skins dos personagens, algo que não fazia parte das atividades previstas pela delimitação de projeto.

Como a adaptação foi feita apenas por uma pessoa, para prosseguir futuramente com esse projeto, sugere-se que um estudo mais aprofundado sobre as possibilidades da *engine* seja realizado. Para que as atribuições de cada artista sejam realizadas com mais eficiência num trabalho em equipe.

Para projetos futuros que aprendam a utilizar a UE, ou outra engine de outro jogo qualquer, para renderizar animações, vale ressaltar a importância de o artista ter um conhecimento básico de como preparar *assets*, como modelos, para uma engine de jogo de forma a minimizar os possíveis problemas causados durante a importação.

REFERÊNCIAS

AKENINE-MÖLLER, Tomas; HAINES, Eric; HOFFMAN, Naty. **Real-Time Rendering:** Third Edition. Boca Raton: CRC Press, 2008.

ALEMBIC. **Introduction.** Disponível em: <http://www.alembic.io>. Acesso em: 04 jun. 2018.

AMID, Amid. **PSOFT Pencil+ Plugin: How 'Your Name' Made Its CG Elements Look Like 2D Artwork.** Disponível em: <http://www.cartoonbrew.com/tools/psoft-pencil-plugin-name-made-cg-elements-look-like-2d-artwork-150511.html>. Acesso em: 12 nov. 2017.

AUTODESK. **TurboSmooth Modifier.** Disponível em: <https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/3DSMax/files/GUID-EA8FF838-B197-4565-9A85-71CE93DA4F68-htm.html>. Acesso em: 04 jun. 2018.

_____. **Smoothing Groups.** Disponível em: <https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/3DSMax/files/GUID-1244162D-A063-486C-BD9B-168466F6488B-htm.html>. Acesso em: 04 jun. 2018.

_____. **Learning MAXScript.** Disponível em: <https://help.autodesk.com/view/3DSMAX/2018/ENU/?guid=__files_GUID_4C14F474_CD23_4001_93DF_0F0F9A6025C7_htm>. Acesso em: 28 jun. 2018.

_____. **Material ID.** Disponível em: <https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/GUID-D8EDE0E1-9694-4844-B58B-A8CB0EBD473B-htm.htmll>. Acesso em: 28 jun. 2018.

BARRÉ-BRISEBOIS, Colin. **Finding Next-Gen – Part I – The Need For Robust (and Fast) Global Illumination in Games.**Disponível em: <https://colinbarrebrisebois.com/2015/11/06/finding-next-gen-part-i-

the-need-for-robust-and-fast-global-illumination-in-games/#more-477>. Acesso em: 26 nov. 2017.

BEANE, Andy. **3D Animation Essentials.** Indianapolis: John Wiley & Sons, Inc, 2012.

BOEHS, Gustavo E.; ANDRADE, Wiliam M. de; VIEIRA, Milton L. H.. **Aventuras na Ilha: A Gestão da Adaptação de uma Obra.** 2014. Disponível em: <https://www.researchgate.net/publication/281461394_Aventuras_na_Ilha_A_Gestao_da_Adaptacao_de_uma_Obra>. Acesso em: 28 jun. 2018.

BIRN, Ian. **3dRender.com - Glossary - Color Bleeding.** Disponível em: <http://www.3drender.com/glossary/colorbleeding.htm>. Acesso em: 28 jun. 2018.

FAILES, Ian. **Upcoming Animated Series 'Zafari' Is Being Rendered Completely With The Unreal Game Engine.**Disponível em: <http://www.cartoonbrew.com/tools/upcoming-animated-series-zafari-rendered-completely-unreal-game-engine-153123.html>. Acesso em: 12 nov. 2017.

_____. **How One Animator Is Making His Own CG Series With Unreal Engine.** Disponível em: <http://www.cartoonbrew.com/tools/one-animator-making-cg-series-unreal-engine-153377.html>. Acesso em: 12 nov. 2017.

KING, Darryn. **Unreal Encourages Filmmakers To Use Its Game Engine.** Disponível em: <http://www.cartoonbrew.com/tech/unreal-encourages-filmmakers-to-use-its-game-engine-111708.html>. Acesso em: 12 nov. 2017.

LIU, Chen. **An analysis of the current and future stage of 3D facial animation techniques and systems.** 2009. 143 f. Dissertação (Mestrado) - Curso de Ciência, Simon Fraser University, Burnaby, 2009. Disponível em: <http://summit.sfu.ca/system/files/iritems1/9923/ETD4934.pdf>. Acesso em: 28 jun. 2018.

LLOPIS, Noel. **Optimizing the Content Pipeline.** Disponível em: <http://gamesfromwithin.com/optimizing-the-content-pipeline>. Acesso em: 28 jun. 2018.

MALHOTRA, Priya. **Issues involved in Real-Time Rendering of Virtual Environments.** 2002. 97 f. Tese (Mestrado) - Curso de Arquitetura, College Of Architecture And Urban Studies, Blacksburg, 2002. Disponível em: <http://fileadmin.cs.lth.se/graphics/research/papers/2011/phd_jm/phd_jacob_munkberg.pdf>. Acesso em: 12 nov. 2017.

MULLER, Mathieu. **Mr Carton – The world's first cartoon series MadeWithUnity.** Disponível em: <https://blogs.unity3d.com/2017/02/23/mr-carton-the-worlds-first-cartoon-series-madewithunity/>. Acesso em: 12 nov. 2017.

PASCHALL, Alexander. **Unreal Engine 4.12 Released!** Disponível em: <https://www.unrealengine.com/en-US/blog/unreal-engine-4-12-released>. Acesso em: 04 jun. 2018.

POHL, Brian J.; BRAKENSIEK, Tim; LOMBARDO, Simone. **Fortnite Trailer: Developing a real-time pipeline for a faster workflow**. S.i: Michele Bousquet, 2016. 47 p.

PRIKRYLV, Josh. **CGI for Television: Don't End Up In The Cartoon Graveyard.** Disponível em: <https://www.awn.com/animationworld/cgi-television-dont-end-cartoon-graveyard>. Acesso em: 12 nov. 2017.

UNREAL ENGINE. **Unreal Engine 4 Documentation.** Disponível em: <https://docs.unrealengine.com/latest/INT/>. Acesso em: 26 nov. 2017.

_____. **Color Grading and Filmic Tonemapper.** Disponível em: <https://docs.unrealengine.com/en-us/Engine/Rendering/PostProcessEffects/ColorGrading>. Acesso em: 28 jun. 2018.

_____. **Fbx Best Practices.** Disponível em: <https://docs.unrealengine.com/en-us/Engine/Content/FBX/BestPractices>. Acesso em: 28 jun. 2018.

_____. **Using Fresnel in your Materials.** Disponível em: <https://docs.unrealengine.com/en-us/Engine/Rendering/Materials/HowTo/Fresnel>. Acesso em: 28 jun. 2018.

_____. **Using Subsurface Scattering in Your Materials.** Disponível em: <https://docs.unrealengine.com/en-us/Engine/Rendering/Materials/HowTo/Subsurface_Scattering>. Acesso em: 28 jun. 2018.

_____. **Types of Lights.** Disponível em: <https://docs.unrealengine.com/en-us/Engine/Rendering/LightingAndShadows/LightTypes>. Acesso em: 28 jun. 2018.

_____. **Post Processing in UE4: Cel-Shading | Live Training | Unreal Engine Livestream.** 2018. (1h10m48s) Disponível em: <https://www.youtube.com/watch?v=cQw1CL0xYBE>. Acesso em: 04 jun. 2018.

SEYMOUR, Mike. **A Fortnite at Epic.** Disponível em: <https://www.fxguide.com/featured/a-fortnite-at-epic/>. Acesso em: 04 jun. 2018.

SWEENEY, Tim. **Build for VR in VR.** Disponível em: <https://www.unrealengine.com/en-US/blog/build-for-vr-in-vr>. Acesso em: 04 jun. 2018.

_____. **If you love something, set it free.** Disponível em: <https://www.unrealengine.com/en-US/blog/ue4-is-free>. Acesso em: 04 jun. 2018.

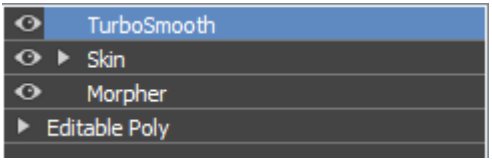
WINDER, Catherine; DOWLATABADI, Zahra. **Producing Animation:** Second Edition. Waltham: Focal Press, 2011.

WRIGHT, Dean. **How Much Does 3D Animation Cost?.** Disponível em: < http://getwrightonit.com/how-much-does-3d-animation-cost/>. Acesso em: 12 nov. 2017.

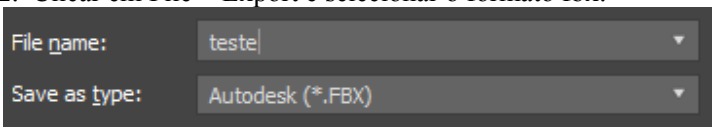
APÊNDICE A – Tutorial das exportações executadas no projeto

Exportando modelo com rig CAT do 3Ds Max para a UE

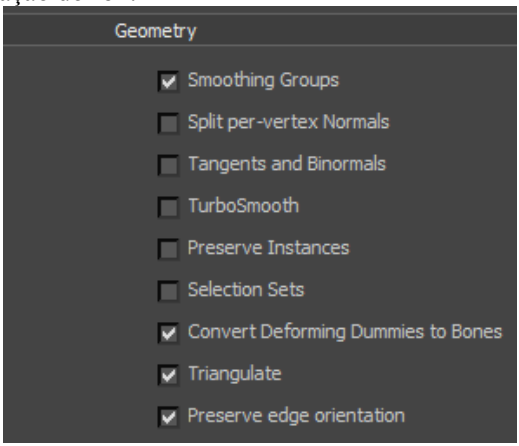
1. Remover ou mover para baixo todos os modificadores que estejam acima do *skin* na lista de modificadores.



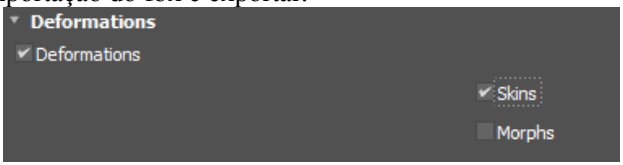
2. Clicar em File > Export e selecionar o formato fbx.



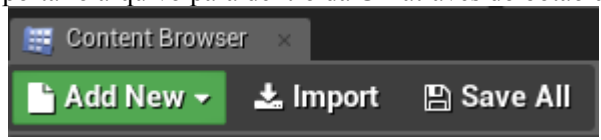
3. Selecionar as opções abaixo na aba geometry do painel de exportação do fbx.



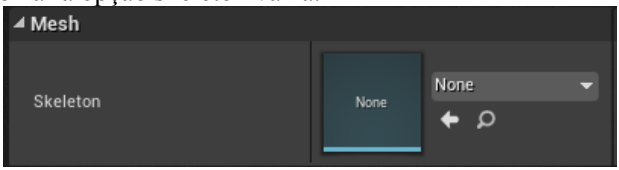
4. Selecionar as opções abaixo na aba animation do painel de exportação do fbx e exportar.



5. Importar o arquivo para dentro da UE através do botão de import.

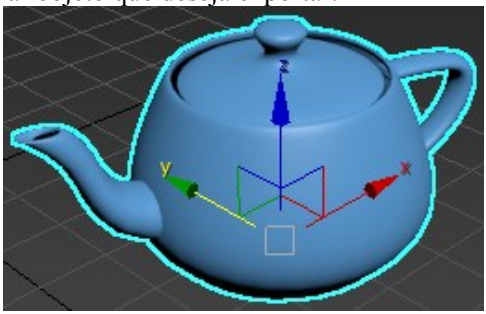


6. Deixar a opção skeleton vazia.

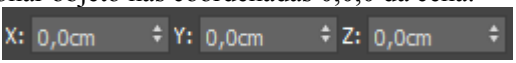


Exportando modelo sem rig do 3Ds Max para a UE

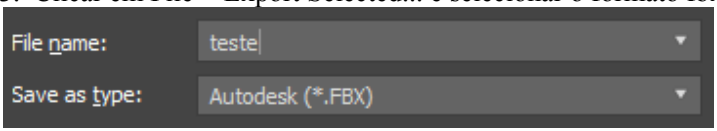
1. Selecionar objeto que deseja exportar.



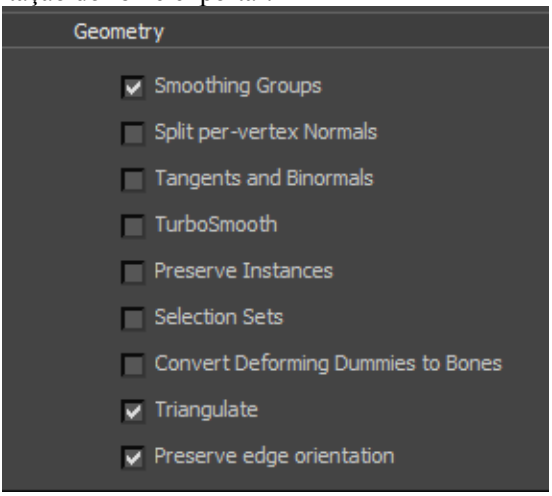
2. Posicionar objeto nas coordenadas 0,0,0 da cena.



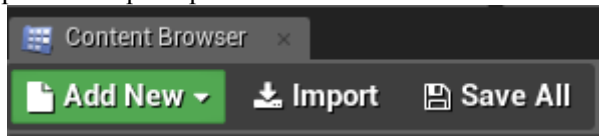
3. Clicar em File > Export Selected... e selecionar o formato fbx.



4. Selecionar as opções abaixo na aba geometry do painel de exportação do fbx e exportar.

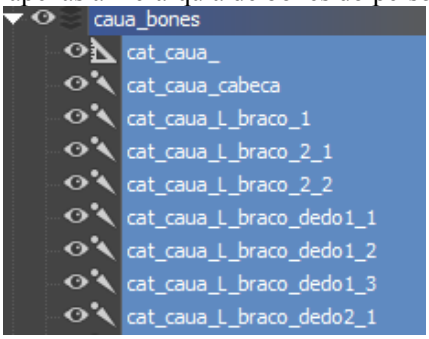


5. Importar o arquivo para dentro da UE através do botão de import.

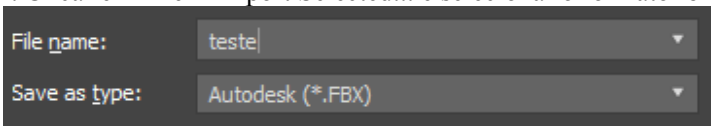


Exportando animação em formato fbx do 3Ds Max para a UE

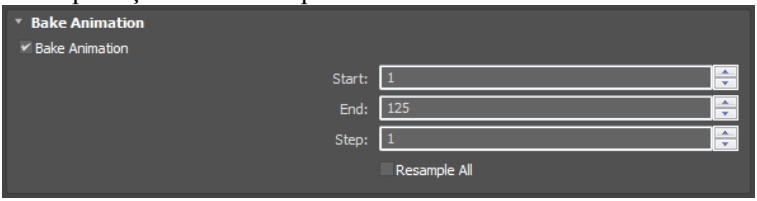
1. Selecionar apenas a hierarquia de bones do personagem.



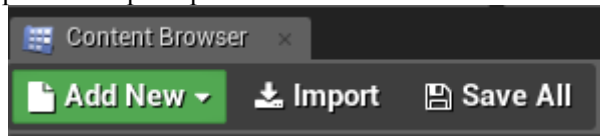
2. Clicar em File > Export Selected... e selecionar o formato fbx.



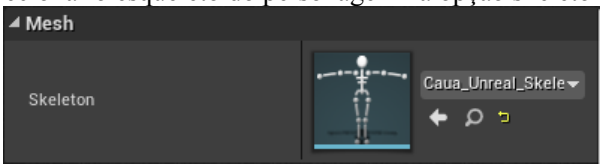
3. Selecionar as opções abaixo na aba animation do painel de exportação do fbx e exportar.



4. Importar o arquivo para dentro da UE através do botão de import.

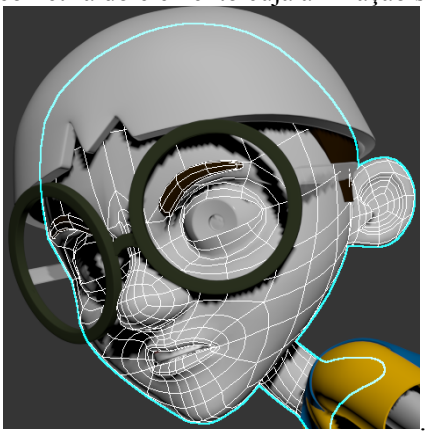


5. Selecionar o esqueleto do personagem na opção skeleton.

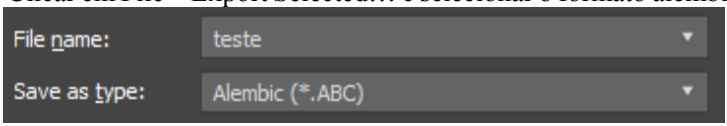


Exportando animação em formato alembic do 3Ds Max para a UE

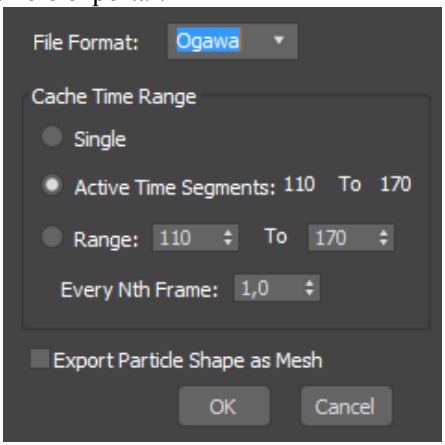
1. Selecionar geometria do elemento cuja animação será exportada.



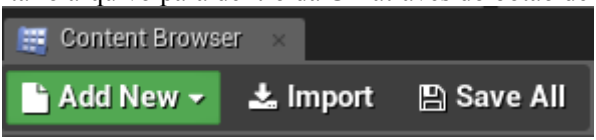
2. Clicar em File > Export Selected… e selecionar o formato alembic.



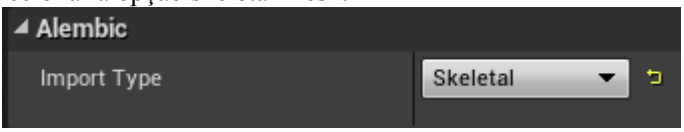
3. Selecionar opções do painel de exportação do alembic conforme imagem abaixo e exportar.



4. Importar o arquivo para dentro da UE através do botão de import.



5. Selecionar a opção skeletal mesh.



6. Selecionar 3Ds Max na aba conversion no menu de importação do alembic.