UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE COMUNICAÇÃO E EXPRESSÃO
DEPARTAMENTO DE GESTÃO, MÍDIAS E TECNOLOGIA
CURSO DE ANIMAÇÃO

SANDRIGO ROSA CAMPOS

DESIGN DE NÍVEIS E ARTE DE AMBIENTES PARA JOGOS 3D

FLORIANÓPOLIS
2025

Sandrigo Rosa Campos

DESIGN DE NÍVEIS E ARTE DE AMBIENTES PARA JOGOS 3D

Trabalho de Conclusão de Curso submetido ao curso de Animação do Centro de Comunicação e Expressão da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Animação.

Orientador: Prof. Gabriel de Souza Prim, Dr.

Florianópolis
2025

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC. Dados inseridos pelo próprio autor.

Campos, Sandrigo Rosa
Design de Níveis e Arte de Ambientes para Jogos 3D / Sandrigo Rosa Campos ; orientador, Gabriel de Souza Prim, 2025.
98 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro de Comunicação e Expressão, Graduação em Animação, Florianópolis, 2025.

Inclui referências.

1. Animação. 2. Jogos digitais. 3. Modelagem 3D. 4. Arte para Jogos. 5. Design de níves. I. Prim, Gabriel de Souza. II. Universidade Federal de Santa Catarina. Graduação em Animação. III. Título.

Sandrigo Rosa Campos

Design de Níveis e Arte de Ambientes para Jogos 3D

Este Trabalho de Conclusão de Curso (TCC) foi julgado adequado para a obtenção de Título de Bacharel em Animação e aprovado em sua forma final pelo Curso de Animação da Universidade Federal de Santa Catarina.

Florianópolis, 4 de Dezembro de 2025.

Prof. Gabriel de Souza Prim, Dr. Coordenador do Curso de Animação UFSC

Banca Examinadora:

Gabriel de Souza Prim, Dr. (Universidade Federal de Santa Catarina)
Nicholas Bruggner Grassi, Dr. (Universidade Federal de Santa Catarina)
Nicolas Canale Romeiro, Me. (Universidade Federal de Santa Catarina)

Documento assinado digitalmente
GABRIEL DE SOUZA PRIM
Data: 12/12/2025 15:28:10-0300
CPF: ***.416.599-**
Verifique as assinaturas em https://v.ufsc.br

Professor Orientador
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Primeiramente, agradeço meu orientador Prim por acompanhar-me durante a jornada de produção deste trabalho, incentivando-me a entregar o melhor de mim, fornecendo ajuda em cada tropeço durante o caminho. As reuniões de segunda-feira pela manhã tornaram-se peça fundamental para o bom andamento do projeto, onde discutíamos (e eu falava demais) os processos da semana anterior e planejávamos os próximos passos. Agradeço por aceitar a complexidade que eu pretendia, por incentivar que a produção iniciasse antes do estritamente necessário, e por impedir que o projeto saísse do controle. Muito obrigado.

Agradeço meu professor Nicholas por me aturar neste último ano de convivência quase diária. As conversas sobre jogos, o *feedback* sobre os processos e as ideias (menção honrosa ao espremedor de limão) são coisas que guardarei com carinho para o resto da minha vida. Muito obrigado.

Agradeço também meu professor Nicolas por fornecer apoio durante este projeto e diversos outros no decorrer do curso. Muito obrigado.

Aos meus amigos Giovanna e Vinícius, por estarem comigo durante todo o processo, ajudando-me de mais maneiras do que seria possível colocar nesta página. À Giovanna, por sofrer comigo. Em certos pontos, pareceu que nossos olhos eram maiores que nossas barrigas, mas, apesar de tudo, conseguimos. Muito obrigado. À Vinícius, por me fornecer apoio tanto psicológico quanto técnico. Suas contribuições permeiam o trabalho como um todo, não apenas as partes que diretamente produziu (ovo frito de dois metros). Tenho muita sorte de ter encontrado alguém com as mesmas aspirações. Muito obrigado.

À meu irmão, Bernardo, por fornecer inspiração para muito do que foi produzido de arquitetônico nesse projeto. Que o resto da tua graduação seja de tanto proveito quanto a minha. Muito obrigado.

À meu pai, Sandrigo, e minha mãe, Taise, por nunca me impedirem de fazer o que eu gosto e me incentivarem a ser a melhor versão de mim. O caminho até aqui não foi o mais linear possível, mas seu apoio em todas as reviravoltas dos últimos dez anos são a principal razão pela qual estou onde estou hoje. Muito obrigado.

À meus animais de estimação, Dora e Julieta, por fornecerem amor incondicional da maneira que só vocês são capazes. Muito obrigado.

E por fim, a todas as pessoas que direta ou indiretamente tenham suas digitais pelo corpo deste trabalho. Ele é um amálgama de tudo aquilo que aconteceu comigo não só nestes últimos quatro anos, mas também um olhar para frente, preparando-me para alcançar meus objetivos. Muito obrigado.

RESUMO

Este artigo buscou explorar as etapas e métodos comuns à produção de cenários para jogos digitais. Abrangeu aspectos como concepção, narrativa, design de níveis e arte de ambientes, além de considerações sobre performance e os requerimentos de renderização em tempo real. Para realizá-lo, primeiramente desenvolve-se um esboço básico dos objetivos da história, usando-o como base para a coleta de referências, fundamentando assim o design do nível; os resultados de tal processo fornecem suporte para o desenvolvimento de metas gerais, como tema e ritmo, auxiliando assim a modelagem do cenário e de *props* que trabalham para adicionar credibilidade ao ambiente; subsequentemente, analisou-se a melhor maneira de transferir tais arquivos entre o programa de modelagem e a *engine*; partiu-se então para produção dentro do motor gráfico, que envolveu elaboração de materiais, escultura de terreno e posicionamento de vegetação. Por fim, elaborou-se uma *cutscene* e uma pequena sequência de *gameplay* como forma de visualização da contribuição do que foi produzido para a ambientação geral do espaço.

Palavras-chave: Design de Níveis; Arte de Ambientes; Arte para Jogos.

ABSTRACT

This article sought to explore the common steps and methods involved in producing environments for digital games. It covered aspects such as conceptualization, narrative, level design and environment art, as well as considerations regarding performance and the requirements of real-time rendering. To accomplish this, a basic outline of the story's objectives is first developed and used as a basis for gathering references, thus grounding the level design. The results of said processes are then used to define overall goals, such as theme and pacing, aiding in environment and prop modelling, which work to add credibility to the space; subsequently, the best way to transfer such models between the modelling program and the game engine were analyzed; then production within the game engine was undertaken, which involved the creation of materials and shaders, the sculpting of terrain, and the placement of vegetation. Finally, a cutscene and a short gameplay sequence were created as a way to showcase the contribution of what was produced to the overall atmosphere of the space.

Keywords: Level Design; Environment Art; Game Art.

ÍNDICE DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A indústria de jogos encontra-se em constante expansão. Em 2025, é esperado que 3.6 bilhões de pessoas, ou 61,5% de todos os usuários da internet jogarão algum tipo de jogo, um crescimento de 4,4% se comparado com 2024[1]. Tal crescimento evidencia a necessidade de contratação de funcionários qualificados em ampla gama de disciplinas relacionadas ao desenvolvimento de tais jogos. Com isto em mente, o objetivo deste trabalho é investigar a fatia da linha de produção referente ao design e arte de ambientes, para assim compreender suas etapas e seus processos, por meio da elaboração de um nível jogável.

Foca-se tanto na elaboração de *level design* adequado, quanto no processo de modelagem, texturização e iluminação. Fez-se uso de ampla gama de referências arquitetônicas, de jogos e de filmes que de uma maneira ou de outra adequam-se aos objetivos artísticos e narrativos definidos durante a pré-produção. Desta maneira, foi possível discutir diversos processos de maneira holística, desde sua concepção em *software* de modelagem, neste caso, *Blender*[2], até sua implementação em motor gráfico, neste caso, *Unreal Engine 5.6*.

No decorrer deste trabalho, abordaram-se as seguintes etapas da produção de cenários para jogos: pré-produção, compreendendo design de narrativa, definição de objetivos e aquisição de referências (3.1); produção no *software Blender*, compreendendo *level design*, iluminação preliminar, *blockout* e modelagem tanto de cenário quanto de *props* (3.2); transferência de arquivos entre o *Blender* e a *Unreal Engine 5.6* (3.3); e produção no motor gráfico *Unreal Engine 5*, compreendendo considerações sobre *performance*; texturização; desenvolvimento de materiais; escultura e *shaders* para *landscapes*; vegetação; e gravação de *cutscene* e *gameplay* (3.4).

---

[1] NEWZOO. **Global Games Market Report**. [S.l.]: Newzoo, 2025. 78 p. Disponível em: https://resources.newzoo.com/hubfs/Free%20Reports/Games%20Market%20Report%20and%20Forecasts/2025_Newzoo_Free_Global_Games_Market_Report.pdf. Acesso em: 20 nov. 2025.

[2] *Blender* foi escolhido para a elaboração deste projeto visto a familiaridade do autor com suas ferramentas.

2 FUNDAMENTAÇÃO TEÓRICA

O processo de elaboração de níveis para jogos 3D divide-se em pré-produção e produção. Elaboram-se primeiramente objetivos narrativos, visuais e de *gameplay*, e buscam-se referências que adequam-se a tais estilos. Para isso, utiliza-se de ampla gama de obras de arte, como por exemplo filmes, literatura, animação, quadrinhos, pinturas, arte conceitual e outros jogos. Também é interessante buscar referências que apenas tangenciam os tópicos selecionados, ou até mesmo que não tenham relação direta com eles, de tal maneira a estimular ideias originais. (DESAMETTI; LAL, 2019).

A etapa de produção caracteriza-se primeiramente por desenvolvimento de *blockout* do cenário, seguido por modelagem arquitetônica e de *props*. Texturizam-se então tais modelos e concomitantemente ilumina-se o nível buscando, na confluência de tais processos, a ambientação correta para as aspirações da narrativa.

De maneira geral, também notou-se a necessidade de tomar cuidado com os requerimentos de performance do projeto, como contagem de polígonos, complexidade de *shaders*, resolução de texturas, gerenciamento de níveis de detalhe e avaliação de técnicas de renderização e seus custos. O uso da *Unreal Engine 5* permitiu maior flexibilidade ao aproveitar de fluxos de trabalho integrados ao *software*, ao invés de ter que ou desenvolver *shaders* ou *geometry nodes* complexos, ou comprar *plugins* para o *Blender* que executariam trabalho similar. Ademais, o fato da renderização ocorrer em tempo real acaba incentivando ajustes mais minuciosos, ao retirar do processo a espera inerente de *ray tracers* do estilo Monte Carlo, como é o caso de *Cycles*, usado pelo *Blender*. Também foi feito uso constante da extensa documentação fornecida pela *Epic Games*, assim como fóruns e vídeos, elucidando peculiaridades e resolvendo problemas encontrados durante a produção.

2.1 *Level design*

Level design (design de níveis) é a disciplina de desenvolvimento responsável pela elaboração das porções ou áreas de um mundo de jogo por onde o jogador deve navegar, e onde exploram-se as regras e mecânicas elaboradas. Sua definição varia, sendo considerado mais ou menos artístico dependendo do estúdio. A parte inicial do processo é chamada de *white boxing*, ou *blockout*, e diz respeito ao processo de posicionamento de volumes simples a fim de definir ritmo e linhas de visão, por exemplo, sem a necessidade de utilização de *assets* finalizados (KARLSSON et al., 2022).

O design de níveis caracteriza-se por sua interdisciplinaridade, encontrando-se no ponto de contato entre áreas diversas do desenvolvimento, como arte, design e programação.

É importante trazer à tona que por vezes existe desconexão entre departamentos responsáveis pela narrativa e departamentos responsáveis pelos níveis (KARLSSON et al., 2022), fazendo com que, em último caso, áreas tenham que ser inteiramente refeitas ou deletadas com base em mudanças na história. Tendo isso em mente, é de grande importância que o design da narrativa encontre-se bem desenvolvido ao começar a etapa de produção, evitando assim retrabalho.

A iluminação pode exercer papel fundamental na etapa de *blockout*, permitindo que os artistas que encontram-se à frente na linha de produção otimizem seus esforços, ao saberem o que será visível e o que não será (KARLSSON et al. 2022).

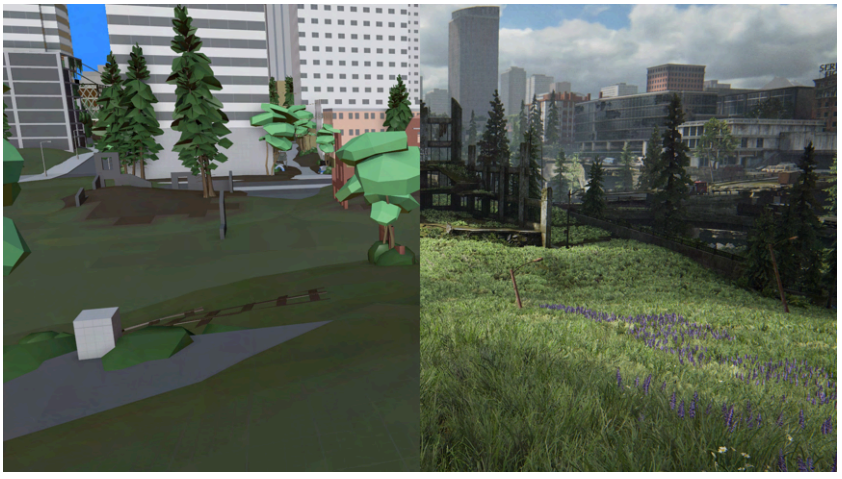
Como exemplo deste processo, é possível citar o trabalho do *game designer* Michael Barclay no jogo *The Last of Us Part II*, de 2020, desenvolvido pelo estúdio *Naughty Dog*. Em artigo publicado em seu blog[3], discute o processo de blocagem do nível "*Ellie Day 1: Downtown*" (Figura 1) em maior detalhe; chama atenção, por exemplo, para a necessidade de constantes iterações para elevar o resultado final a

[3] BARCLAY, Michael. **The Last of Us Part II, Downtown Seattle**. 2021. Disponível em: https://mikebarclay.co.uk/blocktober-2020/. Acesso em: 16 nov. 2025.

mais do que apenas peça de arte de ambientes, fazendo isso por meio da elaboração de interações entre o jogador e o espaço.

Ele detalha o processo específico ao estúdio, que envolve inicialmente o desenvolvimento de um *briefing* curto[4], seguido pela delimitação de "metas de alto nível": tema, exploração de uma cidade pós-apocalíptica mesclada com a melancolia do início de filmes como *Extermínio*[5]; travessia, sensação de liberdade ao poder percorrer o terreno a cavalo, comum à jogos como *Shadow of the Colossus*[6] e *Legend of Zelda: Breath of the Wild*[7]; ritmo, espaço de reflexão narrativa, onde o jogador decide como, quando e o que fazer; e objetivos, entrega do processo de tomada de decisão para as mãos do jogador, evitando linearidade. Passa-se então para o processo de coleta de referências e *blockout* inicial. Aqui, o autor comenta que é preferível manter as formas simples neste período (BARCLAY, 2021), evitando gastar tempo em componentes que possivelmente serão descartados no futuro próximo.

Figura 1: Comparação entre *blockout* e arte finalizada do nível *Ellie Day 1: Downtown*



Fonte: *The Last of Us Part II*, *Naughty Dog*. *Designer*: Michael Barclay[8]

[4] De acordo com o artigo, os designers de nível recebem o *briefing* de outros departamentos do estúdio, como narrativa.
[5] **EXTERMÍNIO**. Direção de Danny Boyle. Roteiro: Alex Garland. Londres: DNA Films, 2002. (113 min.), son., color.
[6] SONY COMPUTER ENTERTAINMENT JAPAN. **Shadow of the Colossus**. [S.l]: Sony Computer Entertainment, 2005. Jogo eletrônico.
[7] NINTENDO EPD. **The Legend of Zelda: Breath of the Wild**. [S.l]: Nintendo, 2017. Jogo eletrônico.
[8] BARCLAY, Michael. **The Last of Us Part II, Downtown Seattle**. 2021. Disponível em: https://mikebarclay.co.uk/blocktober-2020/. Acesso em: 16 nov. 2025.

Menciona também o espaçamento entre pontos de interesse, sendo preferível manter distâncias uniformes para evitar que certas áreas pareçam “mortas” (BARCLAY, 2021). Cita a necessidade de união entre os objetivos da narrativa e do espaço, transformando a área jogável em uma extensão tanto da história das protagonistas, quanto em uma oportunidade de aprofundamento da narrativa contada diretamente pelo ambiente.

2.2 *Environment art*

Environment art (arte de ambientes) é o processo de realização dos *assets* necessários para a arte de um nível, dizendo respeito tanto sobre sua elaboração quanto sobre seu posicionamento no cenário[9]. Diferencia-se de outras áreas da produção de jogos pelo seu foco em composição e legibilidade, sendo peça fundamental da comunicação entre experiência e jogador.

Atribui-se ao artista de ambientes o embelezamento do espaço jogável. Sua definição, assim como a do designer de níveis, varia de estúdio para estúdio, e de produção para produção. Em estúdios independentes, acaba responsável por parte considerável da *pipeline*, como modelagem e texturização, por exemplo, enquanto em estúdios maiores, há um foco em especialização[10] (DESAMETTI; LAL, 2019), à medida que o dinheiro torna-se um problema menor.

É altamente colaborativo, frequentemente entrando em contato com outras disciplinas do desenvolvimento, como animação, arte conceitual, arte técnica, direção de arte, *game design*, *level design* e marketing. Em trabalhos individuais, que possuem estrutura menos rígida, também caracteriza-se por seu tom exploratório, permitindo alterações no design com base em soluções encontradas ao decorrer de qualquer um dos diversos processos (DESAMETTI; LAL, 2019).

---

[9] Ao processo de posicionamento dos assets no cenário, dá-se o nome de *set dressing*, termo proveniente do teatro.

[10] Em tais estúdios, um artista acaba responsável apenas por texturas do cenário, outro para texturas de *props*, outro ainda para modelagem.

3 DESENVOLVIMENTO

Utilizou-se de diversos *softwares* para a execução do projeto: *PureRef*, para compilação de referências e elaboração de *moodboards*; *Blender*, usado para *blockout*, *level design*, modelagem, iluminação inicial, simulação de almofadas e travesseiros, e mapeamento UV; *Marvelous Designer*, usado para a simulação de cortinas e cobertores; *Adobe Substance Painter*, para a criação de texturas; *Adobe Substance Designer*, para a redução no número de texturas utilizadas por meio de *channel packing*; e *Unreal Engine 5*, para desenvolvimento de *shaders*, iluminação final, realização do nível e renderização.

Para a organização do projeto, fez-se uso de uma planilha contendo os prazos (inicial e final) definidos para cada tarefa (Figura 2). Ademais, cada tarefa ainda foi separada em diversas subtarefas de maior ou menor complexidade, permitindo assim amplo controle sobre a produção de cada porção do cenário.

Figura 2: Parte da planilha utilizada para organização do projeto

Cronograma

TÍTULO	Casa
ALUNO	San Rosa Campos

	NÚMERO DA TAREFA	TÍTULO DA TAREFA	DATA DE INÍCIO	DATA DE CONCLUSÃO	DURAÇÃO (DIAS)	% DA TAREFA CONCLUÍDA
1. Pré-Produção	1.1	Definição do Escopo	10/03/2025	14/03/2025	5	100%
	1.2	Montagem do Cronograma	17/03/2025	21/03/2025	5	100 %
	1.3	Blockout Casa	24/03/2025	28/03/2025	5	100 %
	1.4	Blockout Subsolo	31/03/2025	01/04/2025	2	100 %
	1.5	Blockout Garagem	02/04/2025	02/04/2025	1	100 %
	1.6	Blockout Piscina	03/04/2025	03/04/2025	1	100 %
	1.7	Blockout Exterior	04/04/2025	04/04/2025	1	100 %
2. Modelagem	2.1	Modelagem Hall de Entrada	07/04/2025	08/04/2025	2	100 %
	2.2	Modelagem Cozinha	09/04/2025	11/04/2025	3	100 %
	2.3	Modelagem Sala de Estar 01	14/04/2025	16/04/2025	3	100 %
	2.4	Modelagem Sala de Jantar	17/04/2025	18/04/2025	2	100 %
	2.5	Modelagem Escritório	21/04/2025	23/04/2025	3	100 %
	2.6	Modelagem Sala de Estar 02	24/04/2025	25/04/2025	2	100 %
	2.7	Modelagem Banheiros Casa	28/04/2025	02/05/2025	5	100 %
	2.8	Modelagem Quartos	05/05/2025	09/05/2025	5	100 %
	2.9	Modelagem Geral Casa	12/05/2025	16/05/2025	5	100 %
	2.10	Modelagem piscina	19/05/2025	30/05/2025	12	100 %
	2.11	Modelagem garagem	02/06/2025	03/06/2025	2	100 %
	2.12	modelagem escritório subsolo	04/06/2025	06/06/2025	3	100 %
	2.13	modelagem copa	09/06/2025	13/06/2025	5	100 %
	2.14	Modelagem adega	16/06/2025	20/06/2025	5	100 %
	2.15	Modelagem corredor	23/06/2025	25/06/2025	3	100 %
	2.16	Modelagem Banheiro Subsolo	26/06/2025	27/06/2025	2	100 %
	2.17	Modelagem Área para Lixo	30/06/2025	01/07/2025	2	100 %
	2.18	Modelagem vestiário	02/07/2025	04/07/2025	3	100 %
	2.19	Modelagem área de serviço	07/07/2025	08/07/2025	2	100 %
	2.20	Modelagem geral susbolo	09/07/2025	18/07/2025	10	100 %
	2.21	Finalização Modelagem	21/07/2025	04/08/2025	15	100 %
Unreal	3.1	UV Unwrap	01/09/2025	14/09/2025	14	100 %
	3.2	Problemas de Exportação	15/09/2025	21/09/2025	7	100 %
	3.3	Texturização	22/09/2025	21/09/2025	0	100 %
	3.4	Texturas prontas	22/09/2025	28/09/2025	7	100 %
	3.5	Shaders procedurais	29/09/2025	02/10/2025	4	100 %
	3.6	Texturas feitas no Substance	03/10/2025	12/10/2025	10	100 %
	3.7	Environment e vegetação	13/10/2025	16/10/2025	4	100 %
	3.8	Iluminação	17/10/2025	19/10/2025	3	100 %
	3.9	Set dressing final com props prontas	20/10/2025	22/10/2025	3	100 %
	3.10	Ajustes de Personagem e Colisão	23/10/2025	23/10/2025	1	100 %
	3.11	Fazer o Vídeo	24/10/2025	29/10/2025	6	100 %
Texto	4.1	Introdução	30/10/2025	02/11/2025	4	100 %
	4.2	Referencial Teórico	03/11/2025	06/11/2025	4	100 %
	4.3	Desenvolvimento	07/11/2025	12/11/2025	6	100 %
	4.4	Conclusão	13/11/2025	18/11/2025	6	100 %
	4.5	Título e Objetivos	19/11/2025	19/11/2025	1	100 %
	4.6	Resumo, Palavras-Chave e Entregar p/ Orientador	20/11/2025	20/11/2025	1	100 %
	4.7	Pôster	21/11/2025	21/11/2025	1	100 %
	4.8	Entrega do Texto para a Banca	24/11/2025	24/11/2025	1	100 %

Fonte: Produzida pelo autor.

3.1 Pré-produção

A primeira fase do desenvolvimento diz respeito à definição do escopo do que pretende-se produzir, elaborando o esboço narrativo e buscando referências nas mais diversas áreas para assim definir diretrizes a serem seguidas durante as próximas fases.

3.1.1 Design de narrativa

Faz-se necessário desenvolver uma narrativa que sirva de norte para as decisões tomadas durante o período de produção; assim, os componentes do espaço, como *level design*, texturas e iluminação, tornam-se capazes de por si só contarem a história. Nela, assumimos o papel de um jornalista encarregado de entrevistar um famoso bilionário que encontra-se investigado por corrupção. Paulistano, o entrevistado foge da metrópole, onde sente-se perseguido pela investigação em curso, e esconde-se em sua casa de campo localizada em Bom Jardim da Serra, SC.

O voo que levaria o protagonista até Florianópolis atrasa em várias horas; chega à casa apenas durante a madrugada. Encontra a porta entreaberta, e o espaço aparentemente vazio. A partir disso, encarrega-se de investigar as premissas objetivando descobrir o que aconteceu com seus habitantes. Tal investigação levaria-o a explorar a casa em si, seus prédios adicionais (uma piscina e garagem) e o subsolo, até então oculto, espaço reservado ao plantel de funcionários da propriedade.

A progressão seria controlada pela necessidade de encontrar molhos de chave capazes de abrir portas até então fechadas. Como ponto de contato entre o espaço jogável e o mundo exterior, o protagonista possuiria um celular, por onde constantemente conversaria com seu editor; o celular também funcionaria como lanterna para momentos específicos da narrativa.

3.1.2 Definição de metas

Parte-se então para o processo de delimitação das metas do nível, abaixo listadas.

- tema: exploração de um espaço privado, sensação predominante de estar onde não deveria. Junta-se à isso a ambientação isoladora do ambiente ao redor da casa e o uso pontual de luzes para adicionar tensão narrativa, o presságio de que algo de errado aconteceu (ou está prestes a acontecer) naquela casa; toma-se inspiração de filmes como *Parasita*[11], *Ex Machina*[12] e *Nosferatu*[13]; jogos como *Alan Wake II*[14]*, Gone Home*[15], *Vampire: The Masquerade - Bloodlines*[16] e *What Remains of Edith Finch*[17]; e livros, como *Piranesi*[18].
- travessia: estrutura não linear que permite ao jogador explorar a casa como bem entender, revelando pedaços de seu mistério à medida que interage com o espaço. Novamente cita-se jogos como *Gone Home* e *What Remains of Edith Finch* como referências e, tangencialmente, *Blue Prince*[19].
- ritmo: deliberadamente lento, com ênfase na exploração ponderada dos espaços.
- objetivos: após encontrar o incidente incitante da narrativa, abririam-se diversas avenidas de exploração narrativa e espacial. Posicionariam-se documentos e informações de modo a “empurrar” o jogador de volta a linha principal, porém o fazendo de modo a manter amplo o espaço de descoberta.

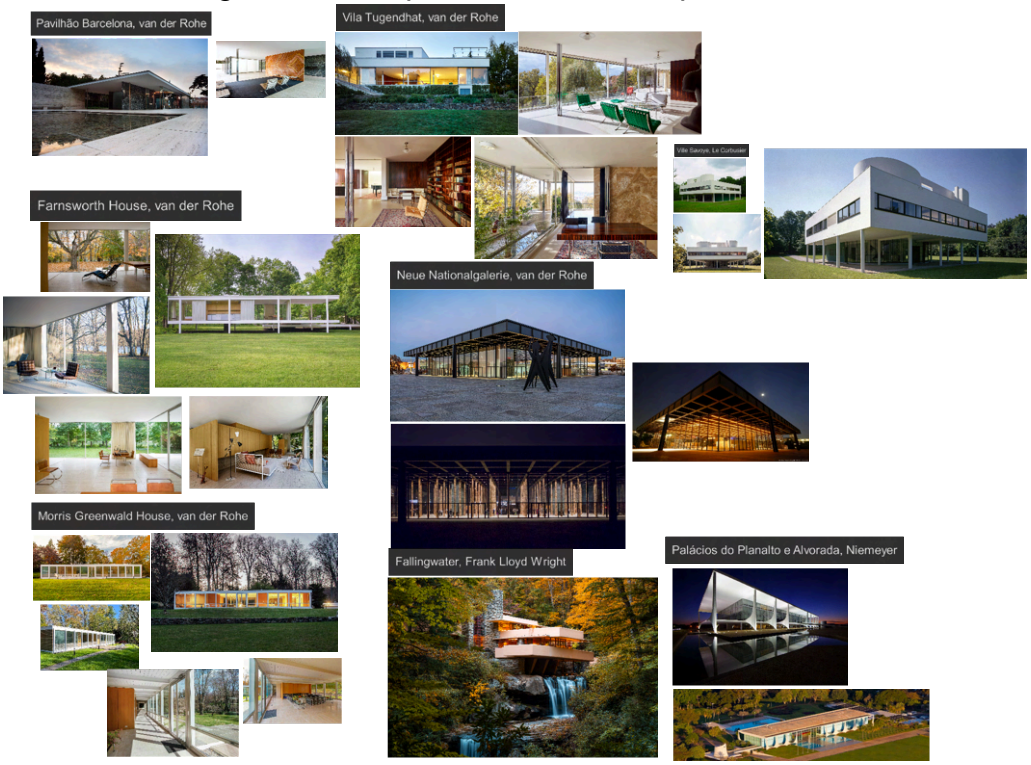
---

[11] **PARASITA**. Direção de Bong Joon Ho. Roteiro de Bong Joon Ho e Han Jin-won. Coreia do Sul: Barunson E&A, 2019. (132 min.), son., color.
[12] **EX MACHINA**. Direção e roteiro de Alex Garland. Reino Unido: Film4, 2014. (108 min.), son., color.
[13] **NOSFERATU**, o Vampiro. Direção de F. W. Murnau. Roteiro: Henrik Galeen. República de Weimar: Prana Film, 1922. (94 min.), P&B.
[14] REMEDY ENTERTAINMENT. **Alan Wake II**. Espoo, FI: Epic Games Publishing, 2023. Jogo eletrônico.
[15] THE FULLBRIGHT COMPANY. **Gone Home**. [S. l.]: The Fullbright Company, 2013. Jogo eletrônico.
[16] TROIKA GAMES. **Vampire: The Masquerade - Bloodlines**. [S. l.]: Activision, 2004. Jogo eletrônico.
[17] GIANT SPARROW. **What Remains of Edith Finch**. [S. l.]: Annapurna Interactive, 2017. Jogo eletrônico.
[18] CLARKE, Susanna. **Piranesi**. Nova Iorque: Bloomsbury Publishing, 2020. 272 p.
[19] DOUGBOMB. **Blue Prince**. [S.l]: Raw Fury, 2025. Jogo eletrônico.

3.1.3 Aquisição de referências

Estipulou-se que a casa teria estilo influenciado pelo modernismo de arquitetos como Ludwig Mies van der Rohe, Le Corbusier, Frank Lloyd Wright e Oscar Niemeyer, como é visto na Figura 3; narrativamente, tal decisão é tomada para criar uma dicotomia entre um movimento arquitetônico predominantemente progressista e os habitantes da casa. Ademais, nota-se em suas obras o uso frequente de superfícies de concreto, como na Praça dos Três Poderes, em Brasília; grandes janelas, na *Edith Farnsworth House*, em Plano, IL, EUA, e na *Neue Nationalgalerie*, em Berlim; e integração com a natureza, como na casa *Fallingwater*, em Stewart, PA, EUA, e na casa *Tirranna*, em New Canaan, CT, EUA. Estes pilares foram considerados fundamentais ao processo de prototipagem e modelagem do espaço. Encontraram-se também diversas outras referências de interiores e exteriores que adequaram-se ao estilo pretendido, fazendo uso do *software PureRef* para compilá-las (Figura 4).

Figura 3: Principais referências arquitetônicas



Fonte: *Moodboard* desenvolvido pelo autor

Figura 4: Referências arquitetônicas compiladas

Fonte: *Moodboard* desenvolvido pelo autor.

Também buscaram-se referências em jogos para áreas como narrativa, direção de arte e renderização (Figura 5). Podem ser citados *Alan Wake II*[20], por sua direção de arte e renderização; *Control*[21], pelo seu uso de concreto; *Hitman World of Assassination*[22], pelo seu *level design*; *Lushfoil Photography Sim*[23], pela sua elaboração do espaço jogável e foco em turismo virtual; *Red Dead Redemption II*[24], pela maneira como renderiza névoa volumétrica; *The Last of Us Part I*[25] e *Part II*[26], pelo uso de técnicas de detalhamento de superfície como *vertex painting*; *Uncharted 4: A Thief's End*[27], pelo posicionamento de props em alguns de seus cenários; e

[20] REMEDY ENTERTAINMENT. **Alan Wake II**. Espoo, FI: Epic Games Publishing, 2023. Jogo eletrônico.
[21] REMEDY ENTERTAINMENT. **Control**. Espoo, FI: 505 Games, 2019. Jogo eletrônico.
[22] IO INTERACTIVE. **Hitman World of Assassination**. Copenhague, DK; Warner Bros., 2022. Jogo eletrônico.
[23] MATT NEWELL. **Lushfoil Photography Sim**. [S.l.]; Annapurna Interactive, 2025. Jogo eletrônico.
[24] ROCKSTAR GAMES. **Red Dead Redemption II**. [S.l.]; Rockstar Games, 2018. Jogo Eletrônico.
[25] NAUGHTY DOG. **The Last of Us Part I**. Santa Mônica, CA, EUA; Sony Interactive Entertainment, 2022. Jogo eletrônico.
[26] NAUGHTY DOG. **The Last of Us Part II**. Santa Mônica, CA, EUA; Sony Interactive Entertainment, 2020. Jogo eletrônico.
[27] NAUGHTY DOG. **Uncharted 4: A Thief's End**. Santa Mônica, CA, EUA; Sony Interactive Entertainment, 2016. Jogo eletrônico.

What Remains of Edith Finch, pela maneira como apresenta sua narrativa e soluciona alguns problemas relacionados ao uso de texturas.

Figura 5: Referências de jogos

Fonte: *Moodboard* desenvolvido pelo autor.

3.2 Produção no *Blender*

Durante o período de realização no *Blender*, vários espaços foram desenvolvidos, existindo diferenciação artística entre eles motivada pela história que pretende-se contar. Abordou-se o processo de maneira exploratória, modelando tentativas e avaliando sua consistência com partes já finalizadas. Várias técnicas distintas de modelagem foram utilizadas, como *hard-surface modelling*, modelagem por curvas, simulações de tecido, simulações de colisão e escultura digital. Fez-se uso frequente de referências de móveis e utensílios, provando-se fundamental na elaboração de malhas verossímeis. Objetivando brevidade, ao discutir os processos minuciosamente, escolhe-se um único exemplo.

3.2.1 *Level design* e iluminação do primeiro andar

Jogos frequentemente alteram a escala de componentes do cenário para, por exemplo, facilitar a locomoção do jogador, visto a adição de camadas de separação entre a ação de pressionar um botão e a ação subsequente de caminhar, por exemplo. Desta maneira, cria-se uma linguagem visual comum à várias experiências, onde ambientes por vezes possuem dimensões por volta de 33% maiores do que seria plausível[28]. Para este projeto, decidiu-se ir em direção contrária a tal tendência, criando maior consistência entre as expectativas impostas à realidade e ao virtual, como é o caso da missão "*Clean House*", do jogo "*Call of Duty: Modern Warfare*", de 2019 (Figura 6). Com isto em mente, os componentes arquitetônicos do nível foram construídos com dimensões idênticas àquelas usadas em construções existentes.

Figura 6: Nível "*Clean House*", *Call of Duty: Modern Warfare,* de 2019

Fonte: *Call of Duty: Modern Warfare*, Infinity Ward. Artista: Jay Miller. *Artstation*[29]

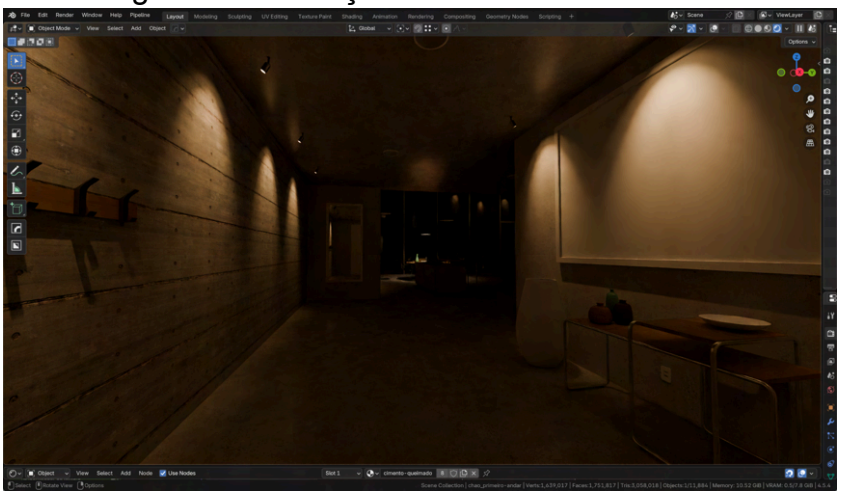
[28] MCMILLAN, Luke. **A Rational Approach To Racing Game Track Design**. In this extensive article, Gamasutra takes an in-depth look at racing game track design, comparing two arcade titles -- *Initial D* and *Maximum Tune* -- and contrasting them, at important points, against the approach used in the *Gran Turismo* series.2011. Disponível em: https://www.gamedeveloper.com/design/a-rational-approach-to-racing-game-track-design. Acesso em: 20 nov. 2025.

[29] MILLER, Jay. **Call of Duty: Modern Warfare - SP - Clean House**. 2020. Disponível em: https://www.artstation.com/artwork/aYNQXX. Acesso em: 18 nov. 2025.

Como significador de opulência, optou-se por pé-direito alto, chegando a sete metros e meio de altura, e espaços comuns amplos, removendo controle sobre a locomoção do jogador. Para manter direcionalidade intencional aos primeiros momentos de *gameplay*, volta-se para o uso pontual da iluminação, guiando o jogador ao incidente incitante da narrativa: a poça de sangue no escritório.

Abaixo, na Figura 7, a iluminação do hall de entrada da casa: indireta nas paredes, com ponto focal à frente.

Figura 7: Iluminação vista do hall de entrada

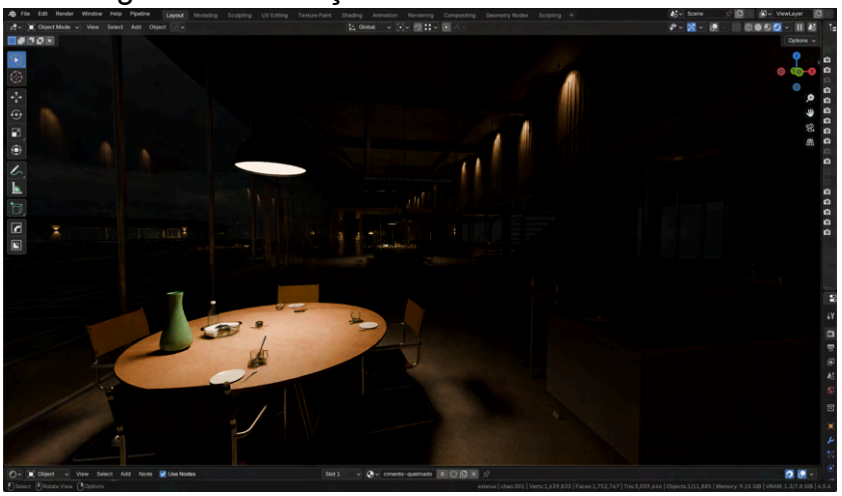


Fonte: Produzida pelo autor.

Preferiu-se uso frequente de iluminação indireta para adicionar formas aos limites da casa, visto que o contrário acaba gerando enquadramentos demasiadamente escuros. A certas paredes também adicionou-se detalhes proeminentes para criar sombras e interações de luz mais complexas, como visto na Figura 8.

A imagem mostra a chegada ao primeiro ponto focal, a mesa da cozinha, onde outro apresenta-se: a mesa de jantar.

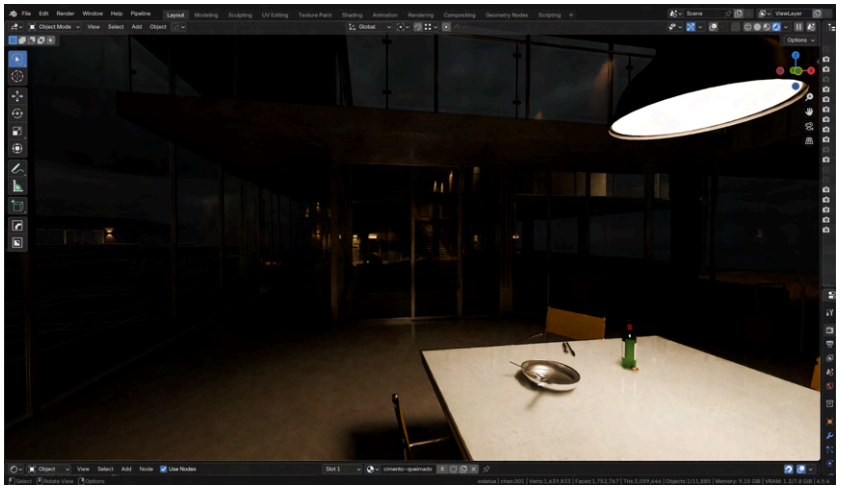
Figura 8: Iluminação vista da mesa da cozinha



Fonte: Produzida pelo autor.

Na Figura 9, é possível observar que ao chegar à mesa de jantar, a iluminação atrai o olhar para a sala.

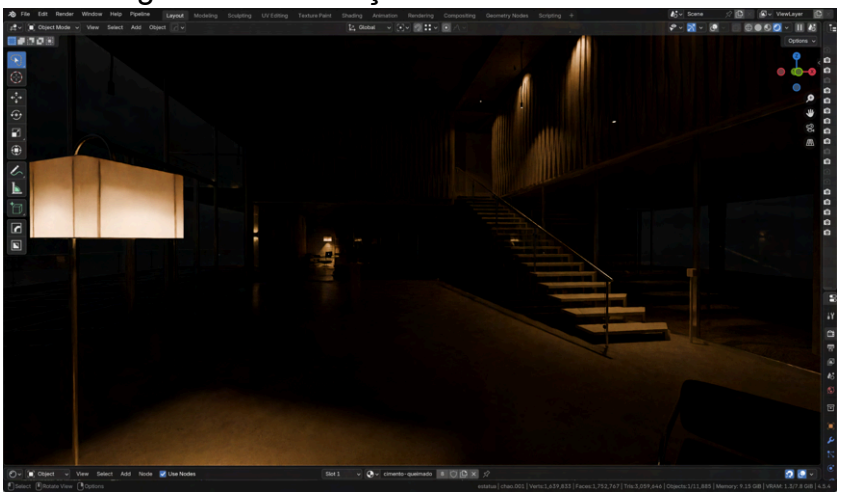
Figura 9: Iluminação vista da mesa de jantar



Fonte: Produzida pelo autor.

Na Figura 10, há separação intencional dos caminhos, ambos possuindo peso: é possível continuar na direção do escritório ou subir em direção à suíte master.

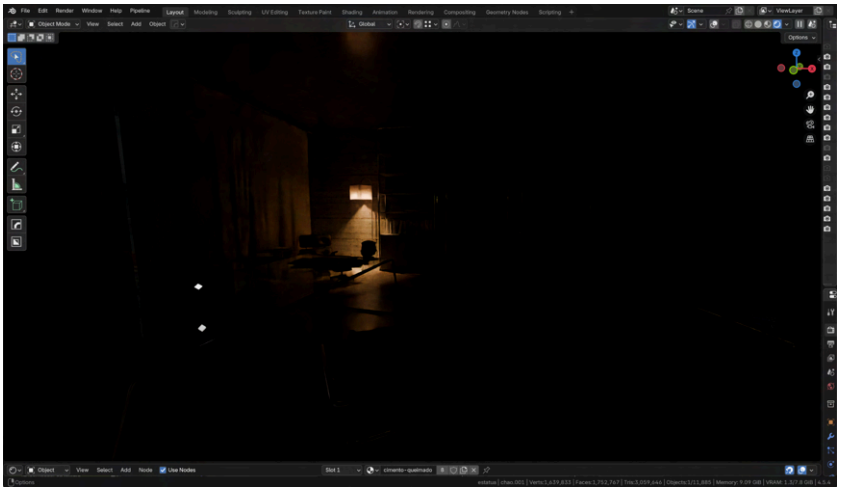
Figura 10: Iluminação vista da sala de estar



Fonte: Produzida pelo autor.

Como é possível ver nas Figuras 11 e 12, ilumina-se o escritório de tal maneira a inicialmente obscurecer o sangue. A narrativa foi planejada de modo que a luz geral da casa desligue assim que aproxima-se deste abajur[30], transicionando para curto período em que toma-se controle do jogador. Tutorializa-se então o funcionamento da lanterna do celular do protagonista, passando o ônus de descoberta da poça de sangue ao jogador.

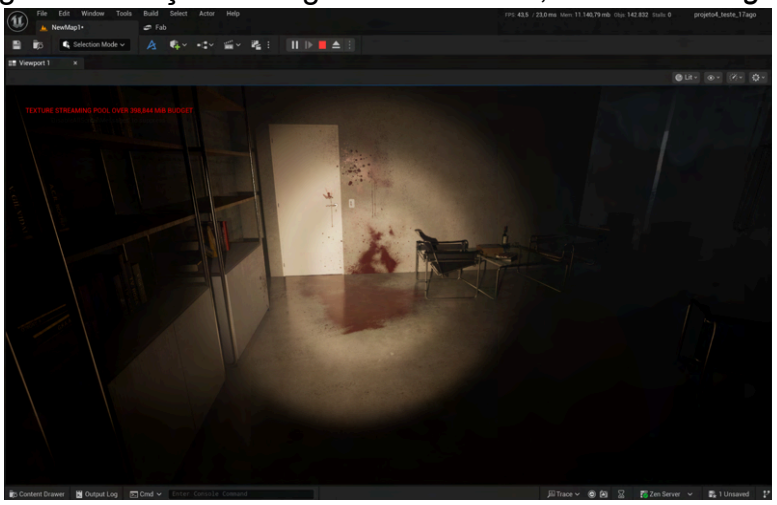
Figura 11: Iluminação do escritório



Fonte: Produzida pelo autor.

[30] Justifica-se tal decisão visto a remoção da casa de áreas urbanas: ocorre ciclagem de seus geradores periodicamente durante a noite, com duração de alguns segundos. Serve narrativamente para gerar momentos de tensão.

Figura 12: Poça de sangue no escritório, na *Unreal Engine*



Fonte: Produzida pelo autor.

3.2.2 *Blockout* e Modelagem da piscina

Ainda no processo de exploração, decidiu-se que a piscina seria posicionada diretamente atrás da casa e encontraria-se em patamar inferior ao do piso principal, para que a mesma não afetasse a vista.

A primeira tentativa (Figura 13) envolveu construção composta exclusivamente de metal e vidro, relembrando uma estufa. Foi descartada considerando a complexidade da modelagem e sua incompatibilidade com o design do prédio principal.

Figura 13: Design inicial da piscina

Fonte: Produzida pelo autor.

As próximas tentativas envolveram construção que utilizaria material idêntico ao usado na casa (Figura 14). Também tomou-se cuidado para que houvesse compatibilidade de formas: o design finalizado é um negativo do design da casa; enquanto o prédio principal possui uma abertura central que permite entrada de luz e faz com que exista declive gradual no telhado em sua direção, a piscina possui centro composto de vidro e elevado em relação aos cantos.

Figura 14: *Blockout* inicial da piscina

Fonte: Produzida pelo autor.

Com *blockout* em estado avançado (Figura 15), aplicam-se ajustes pontuais ao design, como mudança na posição da cozinha. Inicialmente interna, alterou-se para fora do volume principal, alteração esta com motivos enraizados em arte e *gameplay*: considerou-se que exigiria tempo considerável para resolver a junção entre o vidro e a bancada de forma satisfatória, ao mesmo tempo que permitiria contato direto do jogador com área de pouca relevância narrativa.

Figura 15: Atualização no design da piscina

Fonte: Produzida pelo autor.

Com design geral aprovado, passa-se para a finalização de diversos detalhes, que envolvem, por exemplo, a adição de churrasqueira, o posicionamento de *props* variadas, a atualização do design de mesas e cadeiras, e a modelagem de calhas e telhas, como visto na Figura 16.

Figura 16: Design finalizado da piscina

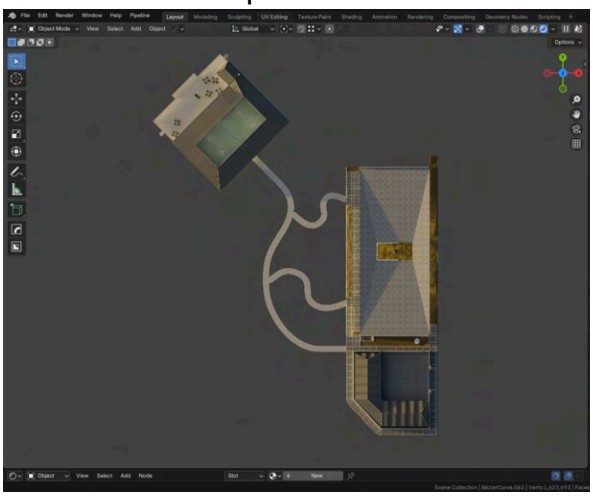
Fonte: Produzida pelo autor.

Após finalizado, considerou-se que o posicionamento da piscina em patamar muito inferior da casa, apesar de servir o propósito discutido inicialmente, não estaria de acordo com o pilar de contato com a natureza, pois dificultaria a arborização do espaço que divide os dois volumes. Também notou-se que tal posicionamento não deixaria claro ao jogador que a área da piscina é explorável.

Decide-se por nova localização, deslocada do prédio principal, porém permitindo o uso de vegetação mais densa.

Para conectá-los, primeiramente explorou-se design orgânico, com curvas suaves (Figura 17). Porém, tal estilo foi considerado em desacordo com o resto do espaço, sendo rapidamente descartado.

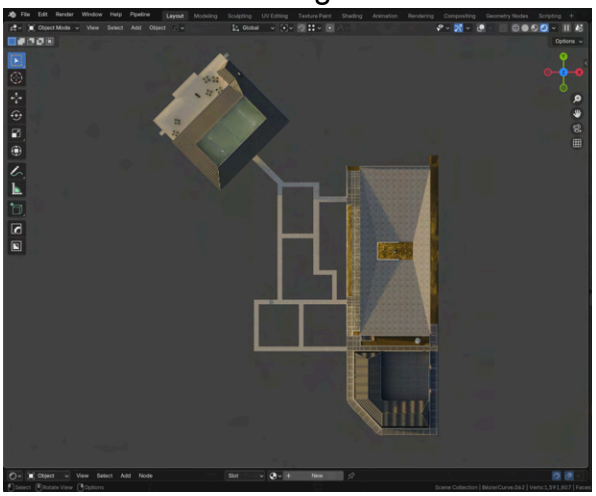
Figura 17: Vista aérea da primeira tentativa de design do caminho entre a casa e a piscina



Fonte: Produzida pelo autor.

Parte-se então para a elaboração de caminho com linguagem de design mais próxima daquela do resto da casa, utilizando principalmente de ângulos retos. O segundo design, visto na Figura 18, provou-se de difícil domínio. Tomou-se pouco cuidado para a elaboração de medidas consistentes, que permitiriam a reutilização de módulos para sua composição, criando a necessidade de elaboração de muitas malhas distintas, sendo descartado por tal motivo.

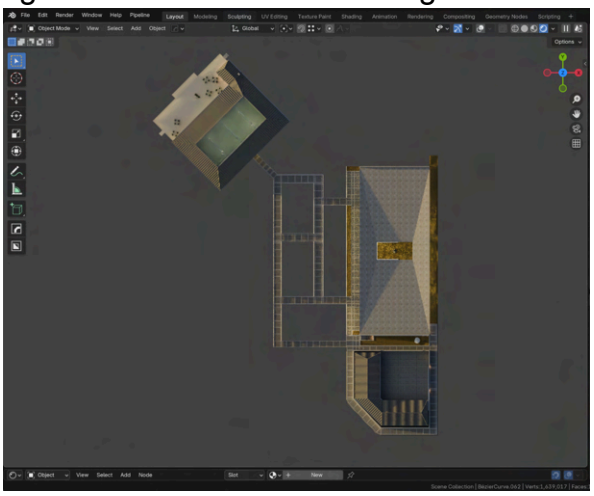
Figura 18: Vista aérea da segunda tentativa de design



Fonte: Produzida pelo autor.

No design finalizado (Figura 19), manteve-se a maioria do caminho no mesmo patamar, concentrando os degraus, que antes causaram problemas, apenas na região mais próxima da piscina. Também modelaram-se módulos que permitiam reuso, reduzindo a quantidade de trabalho necessária.

Figura 19: Vista aérea do design finalizado



Fonte: Produzida pelo autor.

3.3 Transferência de dados entre *Blender* e *Unreal Engine*

Finalizada a modelagem do cenário no *Blender*, muda-se o foco para estudar a maneira mais adequada de transferi-lo integralmente para a *Unreal Engine*. Três opções foram exploradas: as extensões *glTF* e *USD* e o *plugin* "*Blender for Unreal Engine*". Também houve necessidade de solucionar problemas referentes à maneira como o formato de arquivo selecionado gravava as informações.

3.3.1 Formato de Arquivo

A extensão *glTF* (*Graphics Library Transmission Format*), desenvolvida pelo *Khronos Group* para "transferência eficiente de conteúdo 3D através de redes"[31] (tradução nossa), apresentou diversos problemas: por padrão, posicionou os objetos distantes do centro do mundo; todos os objetos têm sua origem no centro do universo, causando problemas quando é necessário alterar sua posição; não aplica modificadores de maneira correta, necessitando aplicação manual em todas as malhas da cena; e gera arquivo consideravelmente maior do que uma das outras alternativas.

O plugin "*Blender for Unreal Engine*"[32] foi inicialmente desenvolvido pela *Epic Games*, embora tenha sido recentemente assumido por desenvolvedores independentes. Apresentou-se inviável visto a complexidade da cena e a maneira como faz a exportação dos objetos: gera um arquivo *.fbx* para cada objeto do cenário, mas o faz de maneira sequencial, repetindo a exportação de materiais e gerando diversas cópias da mesma malha.

Por fim, temos *USD*[33] (*Universal Scene Description*), desenvolvido pela *Pixar* com o intuito de integrar vários formatos de arquivo em um único, mais robusto, permitindo que diversos departamentos de um estúdio trabalhem em cenas ao mesmo tempo. Foi escolhido visto que mantém as hierarquias entre objetos e os

---

[31] KHRONOS GROUP. **glTF - what the duck?**: An overview of the basics of the GL Transmission Format. [20--]. Disponível em: https://www.khronos.org/files/gltf20-reference-guide.pdf. Acesso em: 18 nov. 2025.
[32] LOUX, Xavier. **Blender for Unreal Engine**. Disponível em: https://github.com/xavier150/Blender-For-UnrealEngine-Addons. Acesso em: 19 nov. 2025.
[33] OPENUSD. **Introduction to USD**. 2021. Disponível em: https://openusd.org/release/intro.html. Acesso em: 18 nov. 2025.

pontos de origem; e também por gerar arquivo mais organizado se comparado ao plugin “*Blender for Unreal Engine*”, e consideravelmente mais leve, se comparado à *glTF*: a cena como um todo usando *USD* pesa 142 MB, enquanto apenas a área da piscina usando *glTF* pesa 186 MB.

3.3.2 Problemas com *USD*

Em versões mais recentes do *Blender*, suporte para malhas de colisão do jogador foi retirado do exportador básico de cenas *USD*, fazendo-se necessário ajustar a caixa de colisão individualmente para cada objeto presente no nível ao importá-lo. Para contornar tal problema, utiliza-se um *add-on*[34] que adiciona a informação de colisão, porém o faz utilizando a geometria original do objeto, o que resulta em alto grau de complexidade, sendo então necessário alterar a caixa de colisão de cada uma das malhas para adequá-las aos requerimentos de renderização em tempo real.

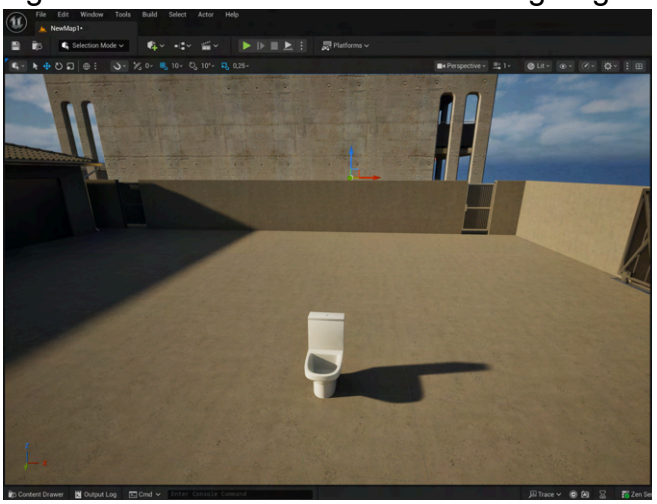
Alguns objetos instanciados passavam por transformação dupla de posição quando exportados: uma considerando sua posição dentro da cena no *Blender*, e outra considerando a transformação de posição sofrida pela “coleção-pai”. Como resultado, certos objetos como vasos sanitários, *props* dos banheiros e as cadeiras usadas na área da piscina acabavam em posições incorretas. Como solução, altera-se a posição da “coleção-pai” para a origem do mundo, fazendo com que não sofra transformação inicial.

Os problemas de transformação acima descritos somaram-se aos problemas de colisão para gerar um terceiro problema: em certos casos, como as malhas dos vasos sanitários e a mesinha de centro da sala do segundo andar, alguns componentes passavam pela transformação incorreta, enquanto outros continuavam onde deveriam. Como o *add-on* para geração de de malhas de colisão de jogador usa uma *bounding box* que contém todos os objetos da pasta que está sendo instanciada, algumas regiões do nível ficavam bloqueadas, como no caso do vaso sanitário do lavabo do primeiro andar: a tampa continuava onde deveria, enquanto o

[34] GERASCH, Lucian. **USDHook4Collisions**. Disponível em: https://github.com/LucianGerasch/USDHook4Collisions. Acesso em: 19 nov. 2025.

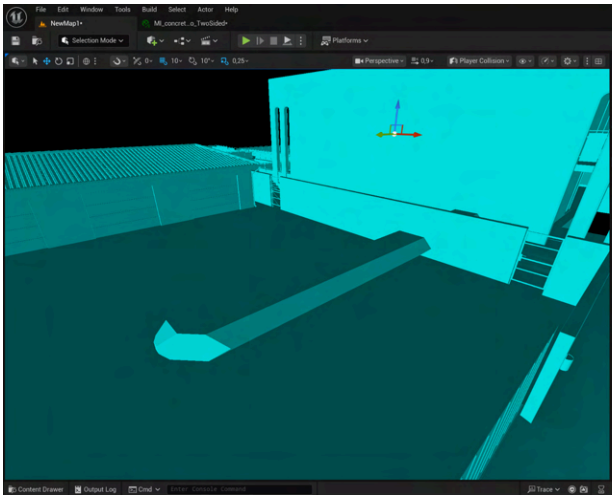
vaso sanitário e a caixa d'água eram "transportados" para o meio da garagem (Figura 20), criando assim uma caixa de colisão que bloqueava o trânsito entre a parte da frente e a parte de trás da casa pelo corredor entre a casa e a garagem (Figura 21). A solução encontrada envolveu transformar coleções instanciadas em objetos reais "linkados" que utilizam da mesma informação de posição de vértices, o que gerou novos problemas.

Figura 20: Vaso sanitário no meio da garagem



Fonte: Produzida pelo autor.

Figura 21: Caixa de colisão do vaso sanitário



Fonte: Produzida pelo autor.

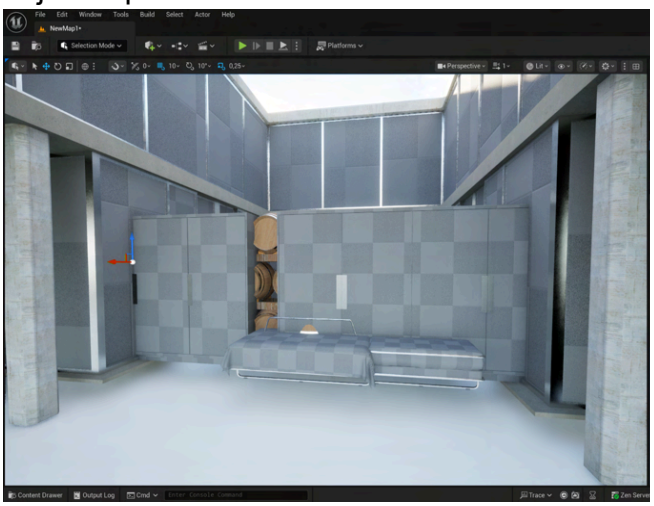
Existem três maneira de replicar um objeto no *Blender*: usando "*Shift+D*", um novo objeto é criado, possuindo propriedades distintas do original; usando "*Alt+D*" (*"copy linked"*), um novo objeto também é criado, porém, crucialmente, as

informações referentes à malha (posição de vértices, normais, *edge weights*) continuam atreladas às do objeto original, possibilitando editar várias cópias ao mesmo tempo; e, por último, também é possível colocar tal objeto dentro de uma pasta e criar uma instância de tal pasta; desta maneira, os objetos copiados são representados por um objeto nulo na visualização da cena, sendo os originais os únicos que realmente existem dentro da memória do programa. Durante o processo de modelagem, tratou-se "*Alt+D*" e instância de coleção como equivalentes, decidindo qual seria utilizado baseado em conveniência. Porém, alguns problemas surgem de tal presunção: primeiramente, ao usar "*Alt+D*", cada cópia do objeto original é tratada com sendo um novo objeto (os vértices são carregados em memória como se fossem de um objeto distinto), acarretando em perda de performance e elevação nos tempos de renderização; segundo, e mais importante para as circunstâncias do projeto, cada cópia é considerada como sendo um objeto distinto durante o processo de criação do arquivo *USD*, fazendo com que edições na *Unreal* que deveriam ser simples tornem-se inviáveis (seria necessário alterar parâmetros individualmente para malhas que podem chegar a ter mais de 1000 cópias na cena). Como solução, deve-se então transformar toda cópia feita com "*Alt+D*" em uma instância de coleção, visto que desta maneira, no arquivo *USD*, apenas um dos objetos idênticos realmente existirá (todos os outros serão apenas referências à *static mesh* original). Porém, apesar de ser possível transformar uma instância em uma "*linked copy*", o *Blender* não possui a funcionalidade de transformar uma "*linked copy*" em uma instância de coleção. Foi necessária a elaboração de um script em *Python* que transformasse cada um desses objetos em instâncias. Como resultado, tornou-se possível editar vários objetos na *Unreal*. O processo em si foi trabalhoso: houve necessidade de checar manualmente todos os objetos presentes na cena para assegurar-se de que aqueles que deveriam estar referenciando a mesma pasta estavam realmente o fazendo.

USD permite escolher qual será o método de seleção de objetos para inclusão no arquivo exportado. Por precaução, optou-se pela opção que segue as configurações de renderização dentro do Blender (para evitar que modificadores escondidos na *viewport* não sejam exportados). Porém, ao fazer isto, certas "coleções-pai", instanciadas várias vezes na cena mas escondidas na *viewport*,

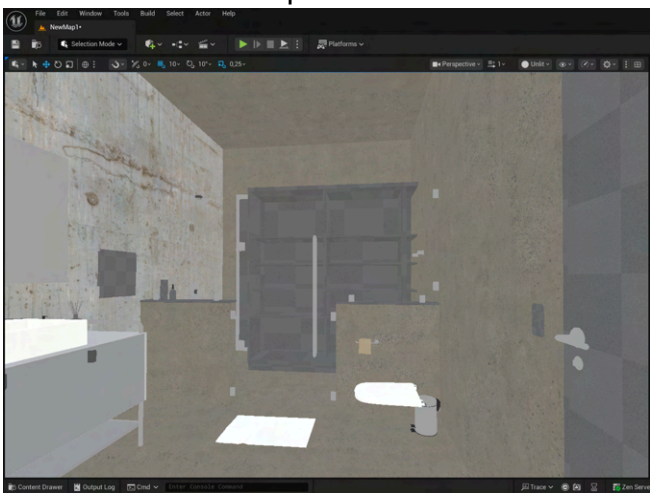
como camas e guarda-roupas, acabavam inclusas no arquivo (Figura 22). Para solucionar tal problema, é preciso manter a visibilidade para o *render* ativa em tais coleções, visto que desativá-la remove as “coleções-filho” da exportação, mas ao mesmo tempo excluí-las do *view layer* (camada de visualização do Blender; define quais objetos estão ativos tanto na *viewport* quanto no *render*). Os guarda-roupas, especificamente, ainda sofriam de outro problema relacionado: quando a “coleção-pai” estava inclusa no *view layer*, as “coleções-filho” passavam por transformação dupla, fazendo com que, por exemplo, o guarda-roupa do quarto de hóspedes acabasse dentro do box do chuveiro do mesmo quarto (Figura 23).

Figura 22: Objetos que não deveriam estar inclusos no arquivo USD



Fonte: Produzida pelo autor.

Figura 23: Guarda-roupa do quarto de hóspedes no box do chuveiro do mesmo quarto.



Fonte: Produzida pelo autor.

Materiais básicos, usando "*Principled BSDF*" e poucos nódulos conectados são transferidos de maneira frequentemente satisfatória. Materiais mais complexos, por sua vez, como o *shader* de vidro desenvolvido para visualização do projeto na sua fase de prototipagem, e que usa de vários nódulos para controlar a resposta de cor na superfície, não são inclusos no arquivo *USD*, fazendo com que objetos que usam tais *shaders* revertam a um genérico quando importados.

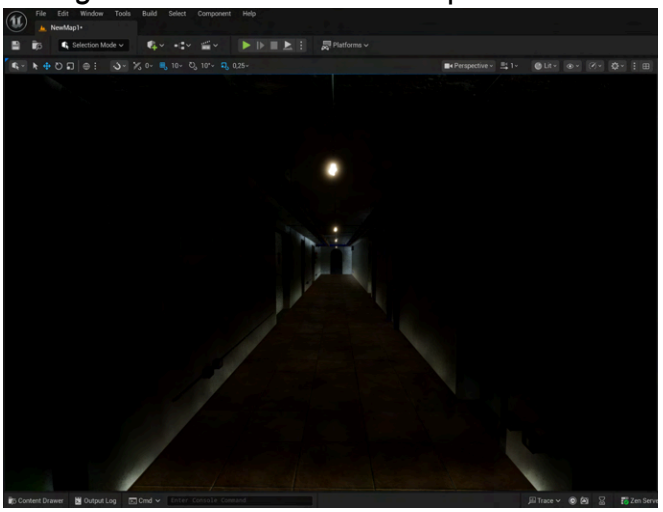
Ao fazer a exportação final, preferiu-se por não exportar texturas e materiais, evitando complexidade indevida. Infelizmente, gerou alguns problemas para atribuição de materiais em instâncias, visto que tornou-se necessário por vezes checar todos os componentes de uma coleção de objetos para aplicar materiais para as partes específicas e, em alguns casos, como nos módulos usados no caminho entre a casa e a piscina, retornar ao Blender para realizar nova exportação, desta vez com materiais aplicados.

3.3.3 *Light Leaking*

A *Epic Games* recomenda[35] que paredes, chãos e tetos que delimitem espaços internos e externos possuam pelo menos dez centímetros de espessura para evitar que a luz ambiente (neste caso, a "luz direcional" da cena) "vaze" para dentro dos ambientes internos. Como a prototipagem aconteceu inteiramente no *Blender*, usando o *ray-tracer Cycles*, que não possui tal requerimento, ao exportar a cena para a *Unreal*, áreas como o subsolo, onde paredes, chão e teto eram apenas planos sem espessura, sofriam de "*light leaking*" (Figura 24). Como solução, retornou-se ao *Blender* (apesar de solucionável dentro da *Unreal*) para modelar uma malha que envelopasse todo o subsolo, bloqueando a luz ambiente.

[35] EPIC GAMES. **Lumen Technical Details**: Dive into the technical details of using Lumen's global illumination and reflections features with software or hardware ray tracing. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-technical-details-in-unreal-engine. Acesso em: 18 nov. 2025.

Figura 24: *Light leaking* no contato entre chão e paredes do corredor do subsolo



Fonte: Produzida pelo autor.

3.4 Produção na *Unreal Engine*

Após a exportação final, trabalhou-se de maneira fluida em diversas áreas da renderização da cena. Simultaneamente, analisou-se a cena como um todo a procura de problemas sejam eles de exportação, ou de modelagem, ou de *UV unwrapping*. Para solucioná-los, criou-se uma cópia sacrificial da cena no *Blender*, e continuou-se com a utilização de *USD* como arquivo de transferência.

3.4.1 Considerações sobre performance

Em primeiro momento, considerou-se trabalhar com a cena assim como ela é exportada, porém rapidamente percebeu-se que tal proposição não estaria de acordo com as necessidades de renderização em tempo real. Considerando isto, é feito uso de ferramentas presentes na *Unreal* que permitem reduzir o custo de renderização da cena como um todo.

3.4.1.1 Vidros

A primeira grande alteração diz respeito ao uso de vidro na composição da casa. Ainda no *Blender*, todas as paredes externas, salvo as laterais, e a parte de baixo de todos os corrimãos internos seria renderizada com tal material. Porém,

testes preliminares na *Unreal* apontaram para redução considerável em performance caso não fosse feita alteração no design. Isto acontece devido a maneira como placas de vídeo calculam camadas de translucência: ordena-as da mais distante à mais próxima da câmera ("*depth testing*"), sendo que cada nova camada de material translúcido faz com que tudo que esteja atrás dela seja renderizada novamente, caracterizando "*quad overdraw*"[36] (quando o mesmo quadrante de 2x2 pixels é renderizado mais de uma vez; exacerbado por sobreposição de camadas translúcidas, mas também por triângulos ou excessivamente finos ou excessivamente pequenos, que ocupam mal os quadrantes). Inicialmente, a casa chegava a contar com oito camadas de sobreposição, fazendo com que tais pixels demorassem muito para serem renderizados. Como solução, abandonou-se o design inicial dos corrimãos, composto principalmente por vidro por outro composto de tábuas de madeira, e alterou-se algumas paredes de vidro para concreto. Desta forma, diminuiu-se de oito para quatro camadas de sobreposição.

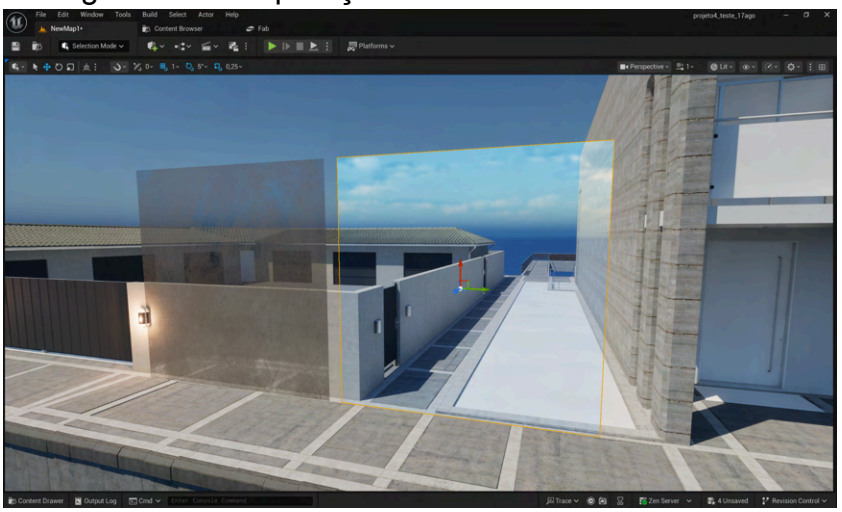
Ainda sobre vidros, estudaram-se maneiras distintas de programação do *shader*. Primeiramente, testou-se a possibilidade de conseguir resultados aceitáveis com técnicas baratas de pós-processamento, que alteram a cor dos pixels após a imagem já ter sido renderizada (por adição ou por modulação). Porém, provou-se desafiante conseguir adicionar variações à superfície, como sujeira e mapa de normal, preferindo-se então por também usar uma técnica com maior custo para performance, porém com maior flexibilidade.

Na Figura 25, à esquerda, vidro utilizando a técnica mais cara, *Default lit* com modo de mesclagem *Translucent*. Nele, é possível adicionar camada de sujeira a apenas uma das direções do plano, permitindo, por exemplo, que o lado externo de uma janela pareça sujo enquanto o lado interno permanece limpo. À direita, vidro utilizando a técnica mais barata, *Unlit* com modo de mesclagem *Modulate*. Como dito anteriormente, é aplicado a cena após o resto já ter sido renderizado. Para os reflexos, usa um *cubemap*[37] simples que pode também usar das informações da

[36] NVIDIA. **Life of a triangle - NVIDIA's logical pipeline**. [20--]. Disponível em: https://developer.nvidia.com/content/life-triangle-nvidias-logical-pipeline. Acesso em: 18 nov. 2025.
[37] EPIC GAMES. **Cubemaps**: The landing page for cubemaps: creation, export, import, and usage within Unreal Engine 4. [201-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/cubemaps?application_version=4.27. Acesso em: 18 nov. 2025.

cena por meio de sondas de reflexo. Considerando as necessidades de performance, as duas técnicas foram utilizadas em conjunto: a primeira para áreas que aparecerão em primeiro plano no vídeo (janelas da casa) e a segunda para janelas à distância (área da piscina).

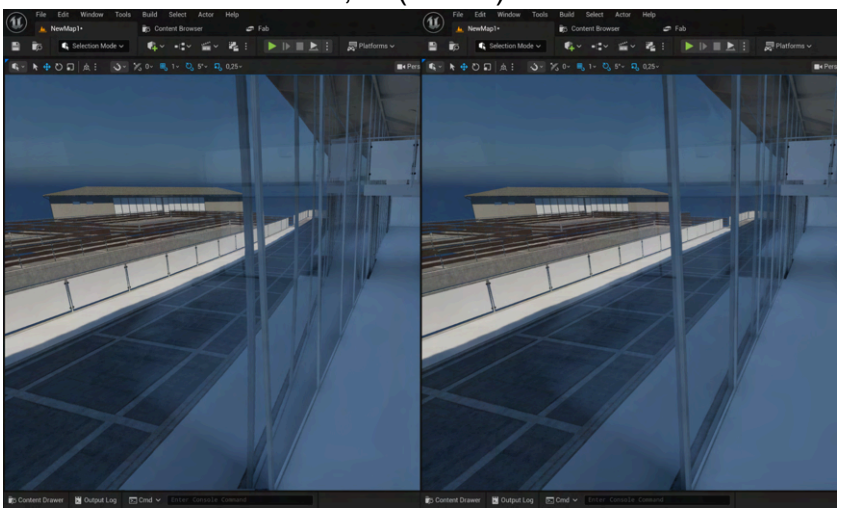
Figura 25: Comparação entre dois *shaders* de vidro



Fonte: Produzida pelo autor.

Sobre o *shader* mais complexo, é necessário utilizar índice de refração impreciso para evitar alguns artefatos visuais decorrentes da maneira como tal índice de refração é calculado: para simular o efeito de “dobra” na luz quando a mesma passa por superfície com índice de refração diferente do ar, usa-se fresnel como método, aumentando o efeito em ângulos oblíquos em relação à câmera; porém, em superfícies de grande comprimento, como é o caso das janelas da frente e de trás da casa, o ângulo de incidência da câmera na superfície do vidro faz com que áreas oblíquas comecem a “dobrar” a luz demasiadamente. Como solução, reduz-se o índice de refração de 1,52, recomendado pela *Epic Games*, para algum valor mais próximo de 1; neste caso, 1,05, como é visto na Figura 26.

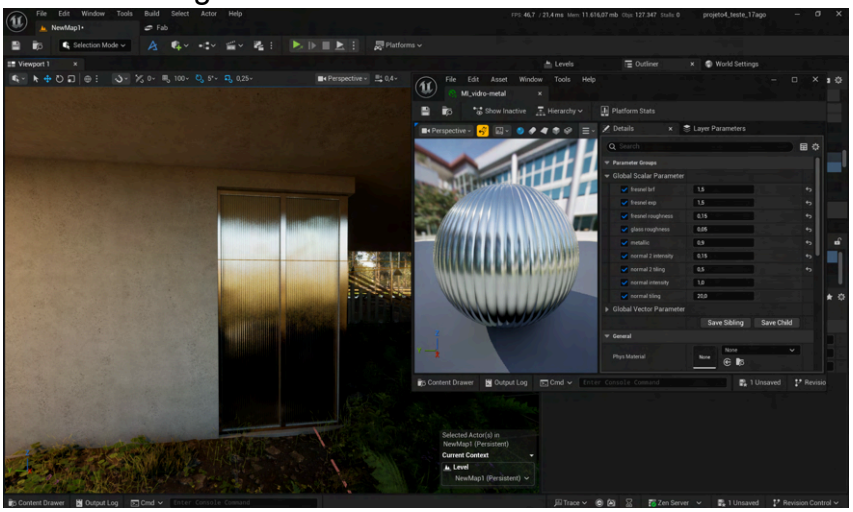
Figura 26: Comparação entre dois valores de índice de refração: 1,52 (esquerda) e 1,05 (direita).



Fonte: Produzida pelo autor.

Pretendia-se também repetir o que havia sido feito no *Blender* com relação a vidros foscos para as janelas dos banheiros. O método encontrado envolvia usar um nódulo próprio do editor de *shaders* da *Unreal* que borra os pixels já renderizados –novamente renderizando vidro após o resto da cena já estar pronto. Porém, o resultado acaba sendo instável e impreciso, gerando artefatos visuais indesejados. Como alternativa, retornou-se ao *Blender* para produzir uma modelagem simples que simulasse a superfície de um vidro canelado, subsequentemente enviando tal modelagem para *Adobe Substance Painter* e lá fazendo um "*bake*" (GIL VIDAL, 2025, p. 51) para gerar um mapa de normal a ser utilizado dentro da *Unreal*. Para salvar em performance, preferiu-se que tal vidro nada tivesse de vidro: ao invés de usar modo de mesclagem que permite translucidez, optou-se por um material opaco com valor metálico igual a 1, assim produzindo reflexos adequados (Figura 27).

Figura 27: *Shader* de vidro canelado



Fonte: Produzida pelo autor.

3.4.1.2 Polígonos renderizados

Outro problema a ser abordado diz respeito à quantidade de polígonos que são renderizados a cada novo quadro. Para reduzi-la, utilizou-se de técnicas diversas de otimização.

Rapidamente percebeu-se que utilizar as malhas assim como eram importadas acarretaria em performance insuficiente. Para evitar isto, usou-se dois métodos: *Nanite*[38], sistema de geometria virtualizada próprio da *Unreal Engine 5*, que permite atualização constante na quantidade de polígonos de cada malha, reduzindo "*quad overdraw*" ao impedir que triângulos menores que um quadrante de 2x2 pixels sejam carregados, e aumentando a quantidade máxima de faces de um objeto, resultando em maior detalhamento; e *LODs*[39] (*Levels of Detail*), aqui gerados dentro da *Unreal*, calculados levando em consideração o tamanho ocupado pela malha no campo de visão da câmera, transicionando entre variações do objeto com quantidades cada vez menores de polígonos à medida que distancia-se dele.

---

[38] EPIC GAMES. **Nanite Virtualized Geometry Overview**: Learn about Nanite's virtualized geometry system and how it achieves pixel scale detail and high object counts. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine. Acesso em: 18 nov. 2025.

[39] EPIC GAMES. **Creating and Using LODs**: How To Create and Use LODs. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-and-using-lods-in-unreal-engine. Acesso em: 18 nov. 2025.

Mesmo com a própria *Epic Games* recomendando *Nanite* para todos os objetos[40], é necessário usar as duas técnicas, visto que tal tecnologia possui uma deficiência: considerando-se que as malhas que a utilizam constantemente alteram a composição interna de triângulos, o método atual para definição de cores de vértices envolve pintá-los diretamente na malha original, ao invés de por instância do objeto, fazendo com que malhas repetidas possuam todas os mesmos valores. Para os objetos que necessitam utilizar tal técnica, prefere-se, então, por gerar *LODs*.

Considerando o que foi dito anteriormente sobre "*quad overdraw*", posicionou-se um volume de modo a englobar a cena; para tal volume é possível definir valores para o tamanho e distância para que malhas que adequem-se a tais parâmetros sejam removidas do nível, reduzindo custo em memória e em tempo de renderização. A isto dá-se o nome de "*distance culling*" (descarte baseado em distância), que soma-se a outras técnicas de descarte[41] já ativadas por padrão:

- *View Frustum culling* (descarte baseado no frustum[42] de visão da câmera): com esta técnica, remove-se da renderização da cena todos os objetos que não encontram-se inclusos no campo de visão da câmera. Usa para isto limites definidos para cada ator presente na cena: uma esfera, usada para detecção de colisão rápida com um teste simples de distância; e uma caixa, mais próxima das dimensões do objeto, produzindo assim resultados mais precisos;
- *Occlusion culling* (descarte baseado em oclusão): soma-se à técnica acima, removendo da renderização os objetos que encontram-se no frustum da câmera mas que estão completamente bloqueados por outro objeto;

---

[40] EPIC GAMES. **Nanite Virtualized Geometry Overview**: Learn about Nanite's virtualized geometry system and how it achieves pixel scale detail and high object counts. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine. Acesso em: 18 nov. 2025.

[41] EPIC GAMES. **Visibility and Occlusion Culling**: An overview of available visibility and occlusion culling methods. [20--]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/visibility-and-occlusion-culling-in-unreal-engine. Acesso em: 18 nov. 2025.

[42] Tronco de pirâmide, neste caso delimitado pelo "sensor" da câmera e sua distância de visualização.

Separou-se também a área jogável em diversos níveis[43], evitando que áreas que não são visíveis durante o recorte de *gameplay* escolhido tenham qualquer "pegada" em memória ou renderização. Para isto, posicionam-se volumes de modo a englobar tais subníveis e testa-se se a cápsula de colisão do jogador está em contato com tal volume; caso a resposta seja positiva, o nível é carregado em memória. Por padrão, a *Unreal Engine* tenta carregar o subnível o mais rápido possível, resultando em perda considerável de performance enquanto os assets são transferidos do armazenamento para a memória do sistema e da placa de vídeo.

3.4.1.3 Sombras

A *Unreal Engine 5* possui dois métodos principais para cálculo de sombras na cena: *ray tracing*[44] ou *Virtual Shadow Maps*[45] (*VSMs*, Mapas de Sombra Virtuais), ambos com vantagens e desvantagens. *Ray Tracing* produz resultados visualmente mais plausíveis, porém apresenta problemas em sua interação com determinadas malhas e materiais. *VSMs*, por sua vez, funcionam gerando uma mapa de sombra de alta resolução (16K, 16.384 x 16.384 pixels), por meio de uma técnica equivalente ao que texturas virtuais (*virtual textures*) fazem com texturas (em ambos os casos, mapas de menor resolução são gerados em tempo real, permitindo que o motor gráfico aloque maior detalhe para áreas que ocupam maior espaço em tela em determinado momento). Soma-se à isso um algoritmo de amostragem chamado de "*Shadow Map Ray Tracing*" (*SMRT*, *Ray Tracing* de Mapa de Sombra), que permite que sombras, ao se distanciarem da base do objeto que as projeta, apresentem-se mais difusas, simulando penumbra. Diferentemente de *ray tracing*,

---

[43] EPIC GAMES. **Level Streaming Overview**: Streaming Levels can be loaded with Level Streaming Volumes or programmatically with Blueprints or C++. [20--]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/level-streaming-overview-in-unreal-engine. Acesso em: 18 nov. 2025.

[44] EPIC GAMES. **Hardware Ray Tracing**: An overview of the features that use hardware to render real-time ray traced results. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-ray-tracing-in-unreal-engine. Acesso em: 18 nov. 2025.

[45] EPIC GAMES. **Virtual Shadow Maps**: Learn about the high-resolution shadow techniques designed with film-quality assets and large dynamically lit open worlds in mind. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/virtual-shadow-maps-in-unreal-engine. Acesso em: 18 nov. 2025.

que avalia raios projetados levando em consideração os triângulos que compõem os objetos afetados por uma fonte luminosa, *SMRT* o faz baseando-se apenas no mapa de sombra gerado pela luz, distribuindo os raios dependendo das dimensões da fonte geradora. Recentemente, uma nova técnica para manuseio de informação de luz direta foi desenvolvida, chamada *MegaLights*[46]. Nela, avalia-se a contribuição de uma fonte luminosa para a cena de maneira estocástica; traça-se um número fixo de raios por pixel, permitindo alto grau de complexidade luminosa (várias fontes luminosas ou fontes luminosas que usam "*light functions*" mais complexas) ao mesmo tempo que reduz-se o custo geral da luz direta, visto que unifica vários passes de renderização em um único. Por padrão, *MegaLights* usa *ray tracing*, evitando o custo de geração de mapas de sombra para cada luz e permitindo que todas as sombras da cena sejam geradas com base no *BVH*[47] (*Bounding Volume Hierarchy*, hierarquia de volumes delimitadores) da cena como um todo, que é produzido apenas uma vez por quadro. Após testes e pesquisa, encontraram-se duas grandes deficiências em como sombras com *ray tracing* são calculadas na Unreal Engine 5: vidros e *Nanite*.

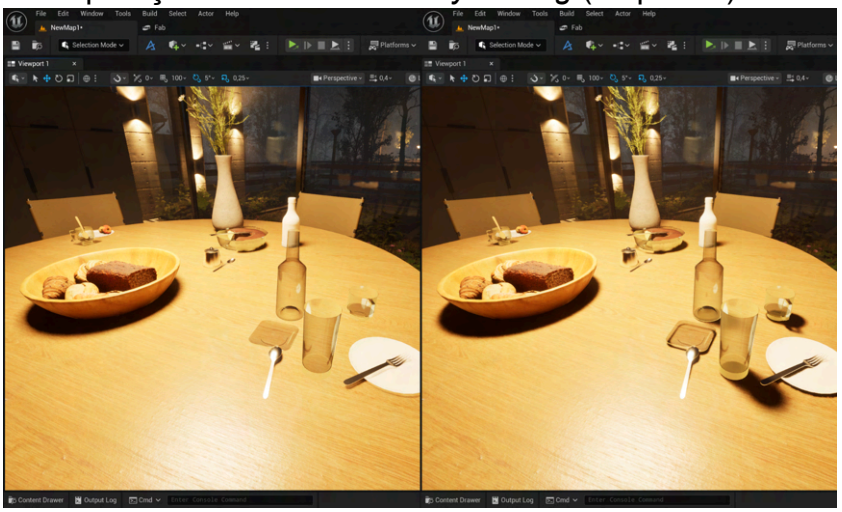
Com relação à vidros, sombras não são geradas para objetos que possuem material com modo de mesclagem configurado para "translucência" (deveria haver uma solução alternativa para tal problema, que envolve ativar a opção na instância do material que diz para o programa renderizar as sombras como se o objeto estivesse usando material opaco, mas tal opção não parece funcionar com *ray tracing*), fazendo com que objetos como copos, taças, garrafas e xícaras pareçam "flutuar" (Figura 28).

---

[46] EPIC GAMES. **MegaLights**: A whole new direct lighting path to place many dynamic and shadowed area lights in a scene. 2025. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/megalights-in-unreal-engine. Acesso em: 18 nov. 2025.

[47] EPIC GAMES. **Ray Tracing Performance Guide**: A selection of topics for improving performance of ray tracing features in your projects. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/ray-tracing-performance-guide-in-unreal-engine. Acesso em: 18 nov. 2025.

Figura 28: Comparação entre sombras *ray tracing* (esquerda) e *VSMs* (direita)

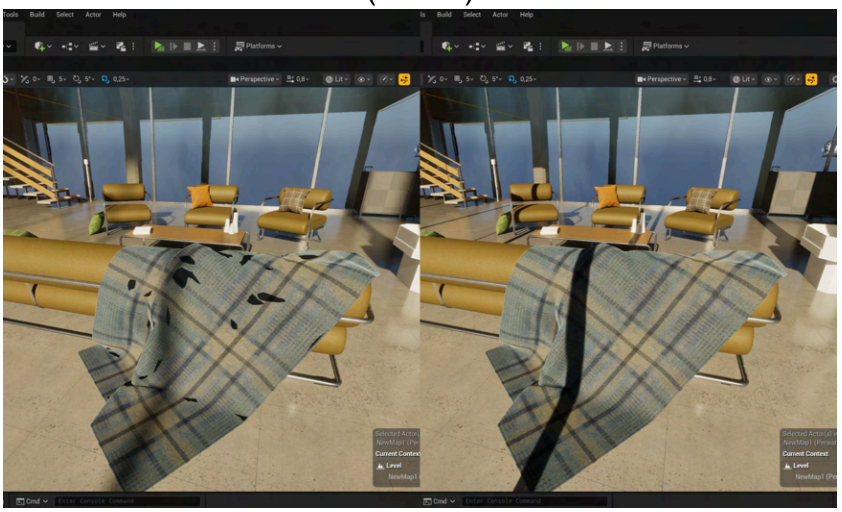


Fonte: Produzida pelo autor.

Com relação à *Nanite*, como as sombras são geradas levando em consideração a *BVH* da cena, que é uma simplificação grossa para evitar altos custos em performance, objetos que usam da técnica de geometria virtualizada apresentam sombras incompatíveis com a complexidade geométrica. Tal deficiência é visível principalmente em *props* com forma mais orgânicas, como cobertores, cortinas e almofadas. Como solução, alterou-se o método de geração de sombras para certas luzes da cena (luz da mesa da cozinha e luz da mesa da sala de jantar, além da luz direcional que representa o sol/a lua), e ativou-se a opção de "*Cast shadow as masked*" no *shader* de certos vidros, para que objetos translúcidos gerem sombras como se fossem opacos.

Na Figura 29, à esquerda, luz direcional com sombra gerada por *ray tracing* produzindo artefatos visuais indesejados em malha com *Nanite*. Alterar o método de geração de sombras para *VSMs*, como visto à direita, elimina os artefatos. Percebe-se também redução no efeito de penumbra, que pode ser emulado por modificação no tamanho do disco luminoso do sol, alteração necessária para geração de sombras mais difusas usando *VSMs*. Após novos testes, parece que tal problema limita-se apenas à luzes direcionais, não sendo reproduzido com *spot*, *rect* ou *point lights*.

Figura 29: Luz direcional com sombras *ray tracing* (esquerda) comparado à *VSMs* (direita)



Fonte: Produzida pelo autor.

3.4.1.4 *Surface Cache*

Uma das simplificações que *Lumen*[48], método de *ray tracing* próprio da *Unreal Engine*, faz com malhas é a geração de "*cards*" com o intuito de representar superfícies de maneira mais eficiente com relação a colisão de raios. Por padrão, doze desses *cards* são gerados e posicionados para cada malha, podendo o número ser elevado até 32 para objetos mais complexos. Superfícies que não possuem cobertura do cache de superfície são representadas como completamente pretas em reflexos com *ray tracing*, perceptível principalmente nas paredes da cena, havendo necessidade de separá-las em objetos menores, de menor complexidade.

Objetos pequenos, por sua vez, são excluídos do cache de superfície para reduzir custo em memória. Para evitar que alguns objetos específicos fossem descartados, aumenta-se a complexidade da cena *Lumen* dentro do volume de pós-processamento que contém o nível como um todo.

É possível alterar o método como reflexos são elaborados pelo motor gráfico de *surface cache* para alguma variação de "*Hit Lighting*", seja apenas para reflexos ou para reflexos e iluminação global. Funciona calculando a iluminação no ponto de

[48] EPIC GAMES. **Lumen Technical Details**: Dive into the technical details of using Lumen's global illumination and reflections features with software or hardware ray tracing. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-technical-details-in-unreal-engine. Acesso em: 18 nov. 2025.

contato entre a malha e o raio. Fazendo isso, torna-se possível controlar o número de "*bounces*" de reflexo na cena, o que permite, por exemplo, que o reflexo do chão de concreto queimado também apresente reflexos. Porém, acarreta em perda de performance, visto que materiais e iluminação são avaliados em cada ponto de impacto. Considerando isso, preferiu-se continuar usando cache de superfície.

3.4.2 Texturização

Dividiu-se em três processos principais:

- bibliotecas de recursos, como *Fab*[49], *textures.com*[50], *PolyHaven*[51] e *Polligon*[52]: texturas prontas, produzidas principalmente por meio de escaneamento de superfícies, gerando assim imagens que encaixam-se bem na linha de produção de renderização *PBR* (*Physically Based Rendering*, renderização baseada em física). *Assets* grandes, como aqueles compostos por concretos, madeiras, metais e tecidos, foram principalmente texturizados desta maneira.
- "*trim sheets*" de autoria própria: envolve a criação de uma única textura a ser utilizada por vários objetos similares. No caso deste projeto, foi usada exclusivamente para a texturização dos livros presentes na cena. Uma textura 4K foi produzida; dividiu-a primeiro em quadrantes, reservando um para a texturização das páginas e segmentando os outros três em doze materiais distintos com resolução de 1K para o *shading* das capas. As lombadas possuem textura individual, aplicada como *decal* nas capas.
- *Adobe Substance Painter*: envolve exportar o modelo do Blender para o *Painter*, organizando camadas e máscaras de modo a criar os canais de textura finais. Usado em objetos onde bibliotecas de texturas não seriam capazes de produzir resultado aceitável, visto a especificidade ou da modelagem, ou da textura necessária. Como exemplo, garrafas de destilados e vinhos, eletrodomésticos, bolo, charuto e receptor de televisão a cabo.

[49] **Fab**. Disponível em: https://fab.com. Acesso em: 18 nov. 2025.
[50] **textures.com**. Disponível em: https://textures.com. Acesso em: 18 nov. 2025.
[51] **Poly Haven**. Disponível em: https://polyhaven.com. Acesso em: 18 nov. 2025.
[52] **Polligon**. Disponível em: https://polligon.com. Acesso em: 18 nov. 2025.

3.4.3 Materiais

Desenvolveram-se diversos materiais que adequam-se às necessidades de cada objeto. Frequentemente usou-se da técnica de instâncias de material, que permite que vários *shaders* dividam a mesma base de código. Dá-se aqui foco principal às técnicas principais utilizadas, explicando funcionalidades e objetivos.

3.4.3.1 *Vertex Painting*

Vertex painting[53], ou pintura de vértices, é uma técnica utilizada para adicionar detalhes a superfícies. Artistas definem valores entre zero e um para os vértices de uma malha, criando assim uma máscara que, quando misturada com uma imagem com informação de profundidade (como *height*, *displacement* ou *bump*), permite interpolar entre duas texturas ou materiais distintos[54]. Cada vértice possui quatro informações de cor, uma para cada canal de uma imagem (R, G, B e alfa), permitindo assim criar misturas entre cinco materiais distintos. Mostra-se útil visto a densidade de *texel*, que define a área de um objeto representada por um pixel de uma imagem. Com tal técnica, é possível texturizar grandes superfícies sem acarretar repetição aparente de textura.

Foi utilizada principalmente para o detalhamento dos concretos presentes na cena. Para tal, desenvolveu-se um *shader* com cinco camadas de material distintas: uma base de concreto; um segundo material de concreto para criar variação; um material de musgo, usado em áreas com pouca exposição ao sol; um material de barro para suavizar a transição entre paredes e chão, principalmente na garagem; e um material que simula umidade, para adicionar poças d'água aos caminhos externos da casa.

Abaixo, na Figura 30, o material de *vertex paint*. Em rosa, o material base; em amarelo, a camada de variação; em vermelho, a interpolação que controla a mistura

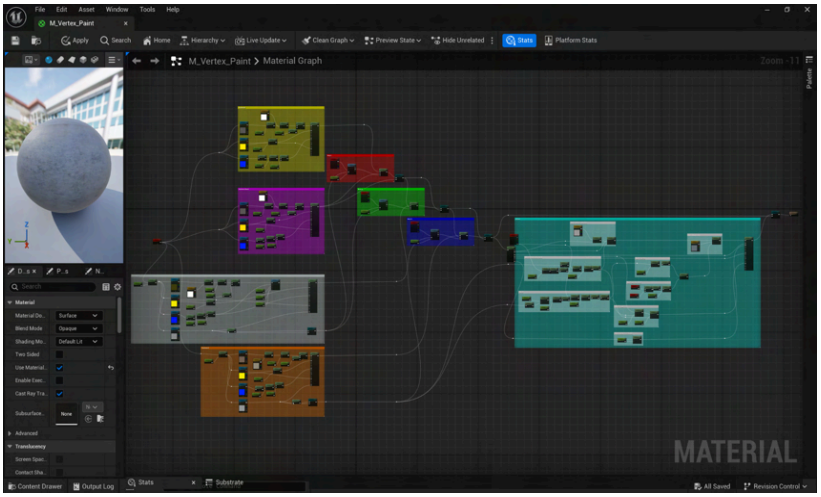
---

[53] EPIC GAMES. **Mesh Paint Mode Tools and Settings**: Descriptions and usages for the various paint tools and settings of the Mesh Paint Mode. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/mesh-paint-tool-reference-in-unreal-engine. Acesso em: 18 nov. 2025.

[54] A mistura ocorre para refinar o resultado da pintura que, considerando a quantidade de vértices das malhas, não possui detalhamento suficiente.

entre estes dois; em branco, o musgo, ocupando mais espaço que os outros pois possui detalhamento adicional ao shader, com adição de aspecto felpudo por meio de *material function*; em verde, a interpolação que controla a mistura anterior com o musgo; em laranja, o barro; em azul, a interpolação que controla a mistura anterior com a sujeira; em turquesa, a umidade.

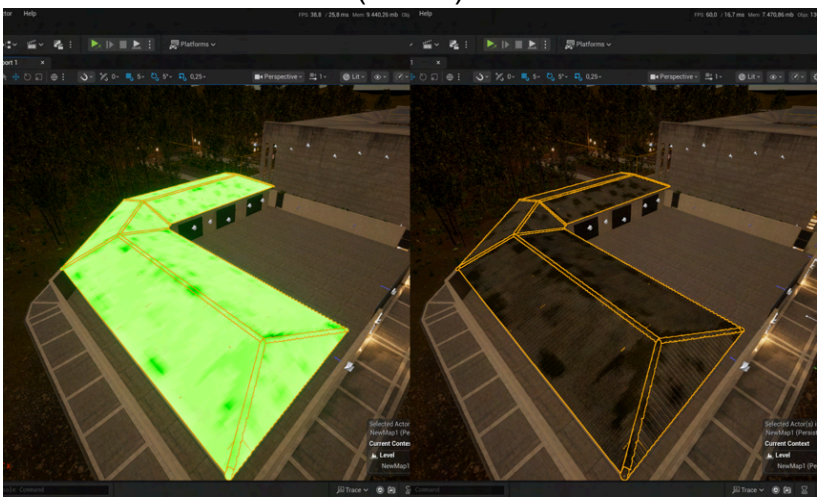
Figura 30: *Shader* de *vertex painting*



Fonte: Produzida pelo autor.

Na Figura 31, o resultado da máscara gerada pela pintura do canal verde dos vértices de algumas malhas. Valores mais escuros indicam áreas onde, neste caso, mais musgo será adicionado ao *shader* final.

Figura 31: Comparação entre a máscara pintada (esquerda) e o material finalizado (direita)



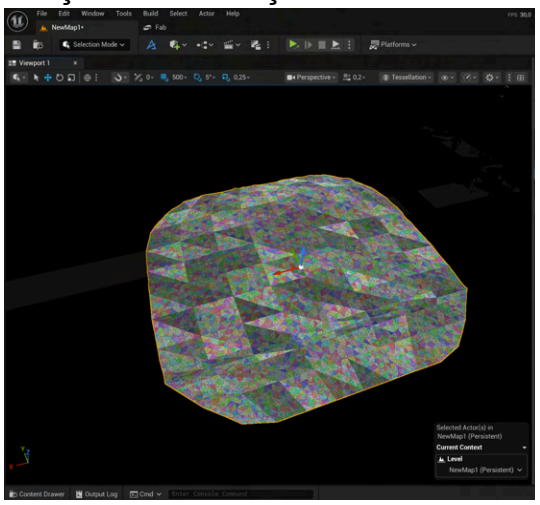
Fonte: Produzida pelo autor.

3.4.3.2 *Nanite tessellation*

Foi necessário, em casos específicos, como apresentado abaixo nas cinzas das lareiras, que a malha exportada do *Blender* tivesse detalhe adicional quando vista em jogo na *Unreal*. Para isto, utiliza-se da técnica de "*tessellation*"[55] (tesselação), aqui usando *Nanite*, onde uma textura de *displacement* controla a posição dos vértices. A malha é subdividida baseando-se na distância do objeto à câmera, adicionando detalhe quanto mais perto se chega (Figura 32).

Tal opção ainda é considerada experimental, sendo necessário adicionar linhas de código ao arquivo de configuração do motor gráfico para permitir seu funcionamento. Feito isso, ativa-se a opção "*Enable tessellation*" dentro do material desejado, e controlam-se parâmetros de amplitude para definir a intensidade do efeito.

Figura 32: Visualização de tesselação na malha das cinzas da lareira



Fonte: Produzida pelo autor.

[55] EPIC GAMES. **Nanite Virtualized Geometry Overview**: Learn about Nanite's virtualized geometry system and how it achieves pixel scale detail and high object counts. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine. Acesso em: 18 nov. 2025.

3.4.3.3 *Fuzzy* e *Subsurface Scattering*

Para os tecidos da cena, combinaram-se duas técnicas que adicionam credibilidade ao *shader*:

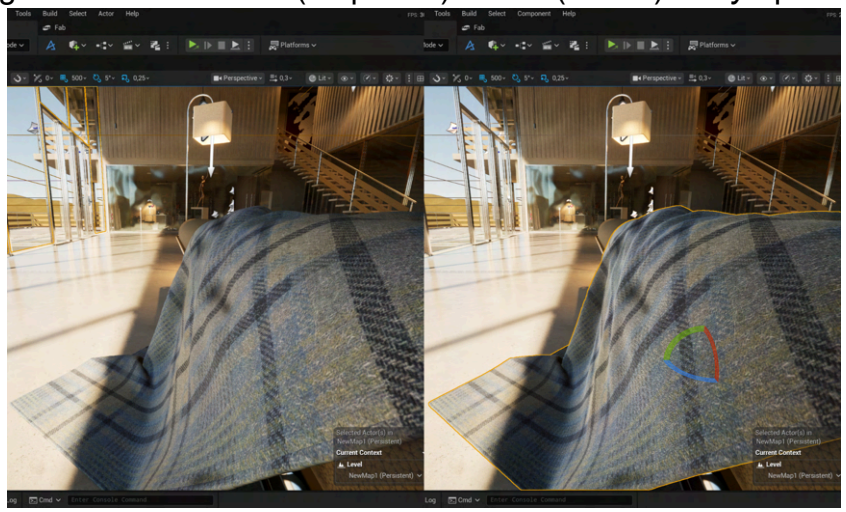
- *Fuzzy*: como dito anteriormente enquanto discutia-se o musgo presente no *shader* de *vertex painting*, existe uma função de material[56] que usa de fresnel para adicionar a aparência de fibras quando o ângulo de incidência da câmera é oblíquo em relação ao material.
- *Subsurface scattering* (*SSS*): usada em diversos objetos orgânicos, como pele e folhas, dando a tais materiais a impressão de profundidade, simulando penetração e dispersão da luz abaixo da superfície.

Para ativar a opção de *SSS*, altera-se o modo de *shading* de *Default lit* para *Subsurface*, o que adiciona novo soquete de saída. Neste caso, controla-se a cor de dispersão multiplicando a cor básica da textura por outra cor qualquer. *Fuzzy*, por sua vez, funciona exclusivamente no canal de cor do *shader*, alterando a luminosidade baseado no ângulo de incidência da câmera na superfície.

Ao adicionar tal *material function* ao *shader*, percebe-se maior luminosidade em áreas oblíquas, como se as fibras do material estivessem “capturando” ou “acumulando” luz, como é possível ver na Figura 33.

[56] *Material Function*, sub-gráfico do editor de *shader* da Unreal que permite definir parâmetros de entrada e de saída, possibilitando implementação rápida de ajustes à materiais.

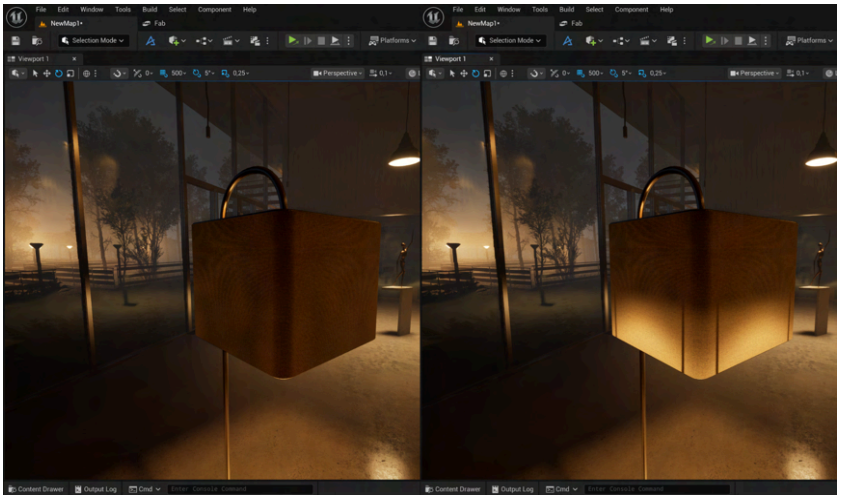
Figura 33: Tecido sem (esquerda) e com (direita) *fuzzy* aplicado



Fonte: Produzida pelo autor.

Na Figura 34, percebe-se que o tecido possui visual demasiadamente opaco caso não altere-se o método de *shading* de seu material.

Figura 34: Comparação entre tecido sem (esquerda) e com (direita) *subsurface scattering*

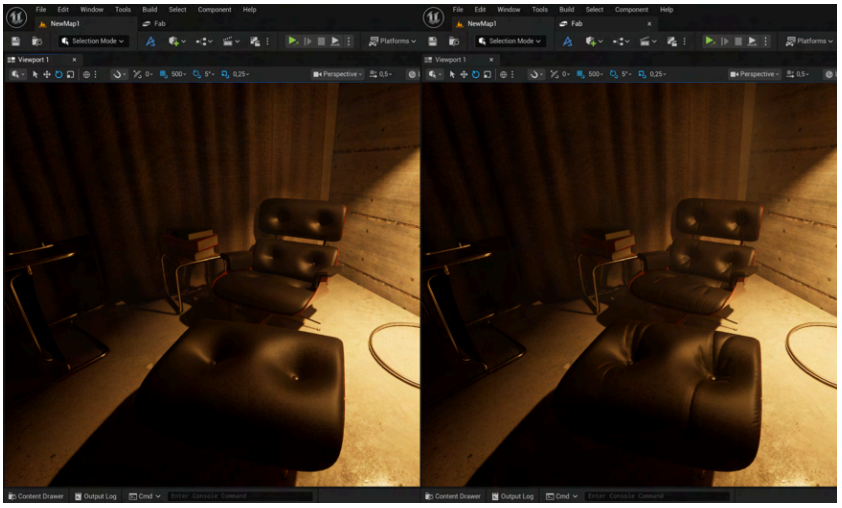


Fonte: Produzida pelo autor.

3.4.3.4 *Detail Normal*

Detail normal[57] é uma técnica empregada para adicionar detalhes à superfícies que usam de texturas repetíveis. Com ela, adiciona-se uma nova textura normal, que é misturada à normal repetível. Neste projeto, foi utilizada no desenvolvimento do *shader* da poltrona do escritório (Figura 35), que usa de um grupo de texturas de couro básico como base, adicionando por cima uma textura produzida por *bake high to low*, a partir de escultura produzida no *Blender*, no *Adobe Substance Painter*. Para mapear tal textura, preferiu-se adicionar um segundo mapa *UV* aos objetos[58].

Figura 35: Comparação entre couro sem (esquerda) e com (direita) *detail normal* aplicada



Fonte: Produzida pelo autor.

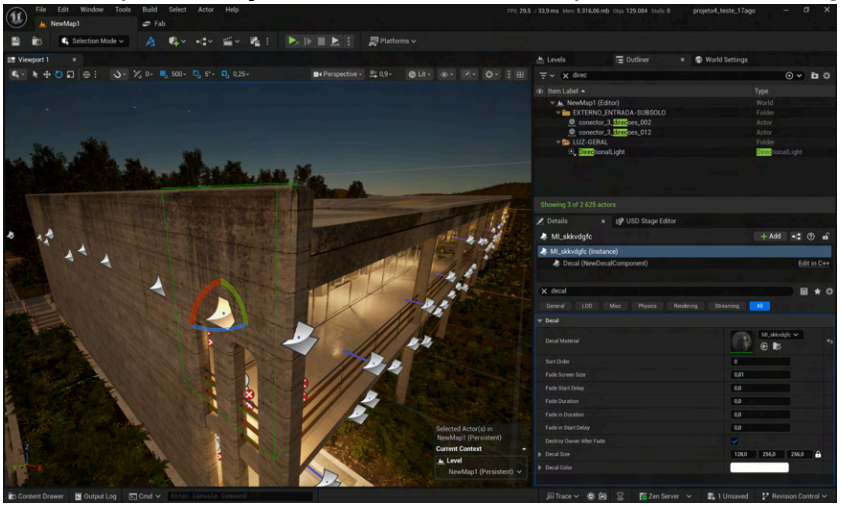
[57] EPIC GAMES. **Adding Detail Textures**: Guide for adding Detail Textures to your Materials. [201-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/adding-detail-textures?application_version=4.27. Acesso em: 18 nov. 2025.

[58] Desta maneira, o primeiro mapa UV controla o comportamento da textura repetível, e o segundo controla o comportamento da *detail normal*.

3.4.3.5 *Decals*

Decals[59] são utilizados para adicionar camada extra de detalhe sobre malhas. Neste projeto, adicionam, por exemplo, sujeira à fachada da casa, dando a ela certo aspecto de descuido. Funcionam definindo uma caixa delimitadora que determina a área de efeito: tudo que intersecta com tal caixa, como visto na Figura 36, terá o *shader* aplicado em sua superfície, a não ser que a opção "*Receives decals*" seja desativada na *static mesh*. É possível controlar a opacidade do efeito distanciando ou aproximando a caixa delimitadora da superfície onde se quer aplicar o *decal*. Preferiu-se por utilizar os *assets* produzidos pela *Quixel* (departamento de escaneamento 3D da *Epic Games*) e disponíveis gratuitamente no *marketplace* da *Unreal Engine*, *Fab*.

Figura 36: Representação dos *decals* na *viewport* da *Unreal Engine 5*



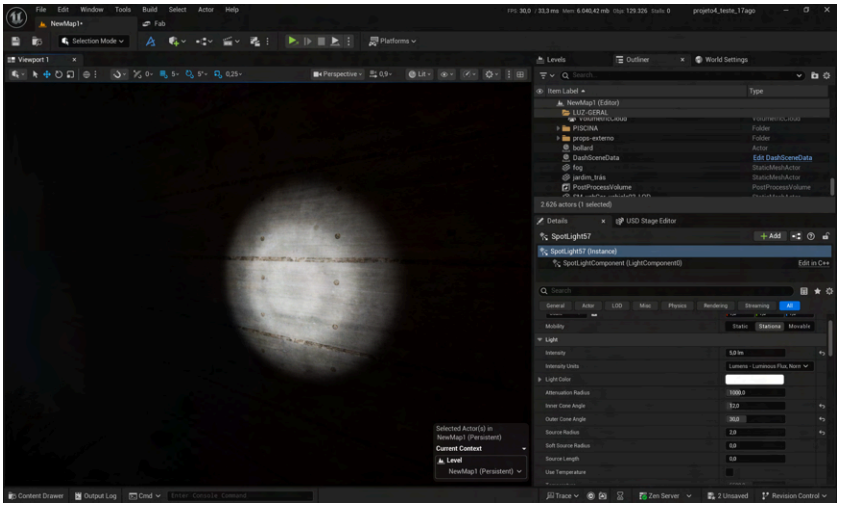
Fonte: Produzida pelo autor.

[59] EPIC GAMES. **Decal Materials**: An overview of decal materials and how to use them in your projects. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/decal-materials-in-unreal-engine. Acesso em: 18 nov. 2025.

3.4.3.6 *Light Functions*

Light functions[60] são materiais que podem ser aplicados à luzes para alterar seu comportamento em cena. Permite, por exemplo, animar o efeito cáustico gerado pela interação entre fonte luminosa e água. Neste projeto, tal material foi utilizado para ajustar o comportamento da lanterna atrelada à câmera do personagem (Figura 37). Para isso, desenvolveu-se uma textura no *Photoshop* e, na *Unreal*, a multiplicou por outra já presente no projeto, adicionando camada de detalhe.

Figura 37: *Spot light* usando a *light function* desenvolvida



Fonte: Produzida pelo autor.

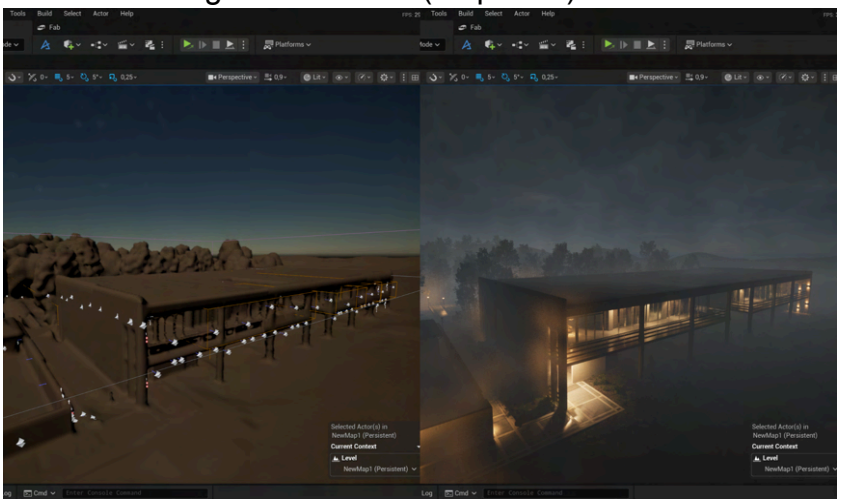
3.4.3.7 Névoa volumétrica

Por padrão, a *Unreal Engine* produz névoa a partir do ator "*Exponential Height Fog*"[61], que cria maior densidade em áreas baixas do relevo. Buscando maior controle sobre o efeito, adicionou-se à tal névoa outra com propriedades volumétricas[62]. Seu *shader* usa de texturas de ruído tridimensionais da própria

[60] EPIC GAMES. **Light Functions**: An overview of using materials to mask a light's projected area. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-light-functions-in-unreal-engine. Acesso em: 18 nov. 2025.
[61] EPIC GAMES. **Exponential Height Fog**: An overview of the height-based, distant fog system. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/exponential-height-fog-in-unreal-engine. Acesso em: 18 nov. 2025.
[62] EPIC GAMES. **Volumetric Fog**: An overview of the volumetric fog and lighting options available with the Exponential Height Fog Component. [202-]. Disponível em:

Unreal Engine para controlar a densidade em cada ponto do volume delimitante. Existem parâmetros para definir a repetição e o deslocamento das texturas de ruído, a fim de simular os efeitos do vento. Adicionou-se também uma opção que faz com que a névoa “grude” às malhas, por meio de avaliação do *distance field*[63] gerado pelo motor gráfico (Figura 38). Define-se uma distância, calculada a partir da superfície do *distance field*, que será utilizada como ponto final para o cálculo da névoa, sendo que o que é delimitado pelo volume criado terá névoa aparente.

Figura 38: *Distance field* global da cena (esquerda) e névoa volumétrica (direita)



Fonte: Produzida pelo autor.

Encontraram-se problemas referentes à maneira como a névoa era atualizada de quadro a quadro, sendo necessário fazer ajustes ao “*DefaultEngine.ini*” do projeto, arquivo responsável por guardar as informações da versão do motor gráfico em uso. À ele foram adicionadas tais linhas:

- r.VolumetricFog.HistoryWeight = 0.667
    - o valor é originalmente configurado para 0.9, dando mais “peso” à informação do quadro anterior. Funciona adequadamente em *gameplay*, mas apresenta problemas quando fontes luminosas atravessam o volume, como é o caso do carro na *cutscene*; áreas não

https://dev.epicgames.com/documentation/en-us/unreal-engine/volumetric-fog-in-unreal-engine. Acesso em: 18 nov. 2025.

[63] EPIC GAMES. **Mesh Distance Fields**: An overview of Mesh Distance Fields and its available features that you can use when developing your games. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/mesh-distance-fields-in-unreal-engine. Acesso em: 18 nov. 2025.

mais iluminadas pelas luzes dianteiras continuavam comportando-se como se ainda estivessem.

- r.VolumetricFog.LightSoftFading = 8
    - originalmente configurado com valor 0; funciona apenas em luzes do tipo retangular ou holofote; aumenta a suavidade da transição entre áreas iluminadas e áreas não iluminadas.
- r.VolumetricFog.UpsampleJitterMultiplier = 2
    - originalmente configurado com valor 1; adiciona um multiplicador ao tremor da névoa que reduz pixelização, porém o faz adicionando ruído à imagem. A troca foi considerada válida, visto que a alternativa, reduzir o tamanho de cada *voxel* (unidade de volume), apresentou-se como inviável devido à seu alto custo em performance.
- r.VolumetricFog.HistoryMissSupersampleCount = 8
    - originalmente configurado com valor 4; aumenta o número de amostras para *voxels* que não possuem informação temporal, ou seja, que estavam sendo descartados, resultando em menor quantidade de artefatos nas bordas do enquadramento quando a câmera movimenta-se rapidamente.

3.4.4 *Landscape*

Após finalização da texturização do cenário produzido no *Blender*, inicia-se a produção do relevo a ser utilizado na cena, desta vez inteiramente dentro da *Unreal Engine*. Inicialmente, estudou-se a possibilidade de utilizar programas adicionais, como *Gaea* ou *World Creator*, para desenvolver mapas de altura (“*heightmaps*”). Tais programas trabalham a base de nódulos e possuem ferramentas robustas de erosão, sendo frequentemente utilizados por estúdios do ramo de jogos. Porém, considerando as condições de iluminação do cenário e o relevo da região que serve de referência, julgou-se desnecessário sair do motor gráfico, preferindo-se usar as ferramentas já embutidas.

O processo de criação do terreno envolve primeiramente a adição de um "*Landscape actor*"[64] à cena usando a aba "*Landscape mode*". Definem-se então alguns parâmetros que controlam a maneira como tal ator será criado:

- "*Section size*": tamanho de seção, usado para *LOD* e *culling*; importante encontrar um balanço entre tamanho e custo para a CPU, visto que seções menores demoram mais para serem avaliadas. Por padrão, 63x63 *quads*;
- "*Sections per component*": seções por componente, define o tamanho de cada subdivisão do "*Landscape actor*"; quanto mais componentes por seção, maior é o custo de avaliação na CPU. Por padrão, 1x1 seção;
- "*Number of components*": número de componentes para cada seção do *landscape*, limitado a 32x32 pois cada componente possui um custo de CPU associado. Por padrão, 8x8 componentes;
- "*Overall Resolution*": resolução geral, define o número de vértices gerados para cada dimensão do *landscape*. Por padrão, 505x505 vértices.

Por precaução, preferiu-se utilizar os parâmetros originais, visto que a *Epic Game*s, na documentação da *Unreal Engine*, diz que é necessário cuidado ao alterar o "*Section size*" pois acréscimo ao número de componentes aumenta tempos de compilação e pode drasticamente afetar a performance.[65]

Com o *landscape actor* criado, parte-se para o processo de modelagem, que envolve utilizar ferramentas não muito diferentes daquelas encontradas em programas como *Blender* e *ZBrush*, escolhendo pincéis para aumentar ou diminuir a elevação. Possui também ferramentas que simulam erosão pluvial e eólica, refinando as formas criadas anteriormente. Durante tal processo, foi feito uso de várias referências de áreas próximas à Serra do Rio do Rastro, em Bom Jardim da Serra, SC, visando replicar seus morros ondulados.

---

[64] EPIC GAMES. **Landscape Quick Start Guide**: Getting up and running with the basics of the Landscape System in Unreal Engine. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/landscape-quick-start-guide-in-unreal-engine. Acesso em: 18 nov. 2025.

[65] EPIC GAMES. **Creating Landscapes**: Guide to creating new Landscape terrains. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-landscapes-in-unreal-engine. Acesso em: 18 nov. 2025.

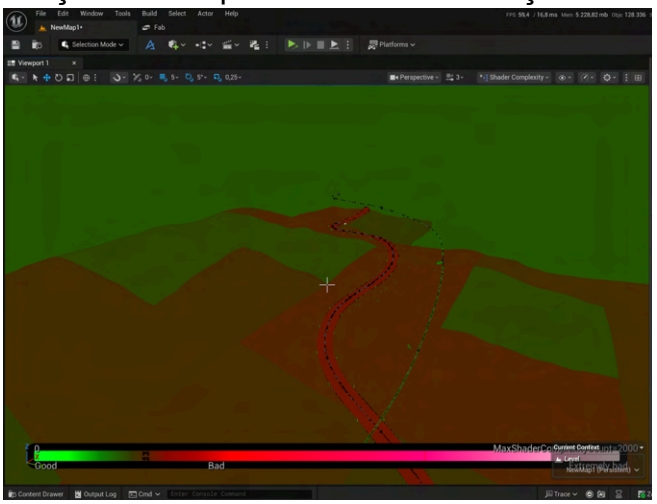
3.4.4.1 *Landscape shader*

Shaders de *landscape*, como aqueles que usam de pintura de vértices, também utilizam de diversas camadas de material para criar variação. Para o projeto, usou-se como base um *shader* desenvolvido por *Unreal Sensei*[66], que possui cinco camadas distintas, sendo possível adicionar até mais três, caso seja necessário. A partir de tal material, definiram-se as texturas a serem utilizadas em cada camada: grama, barro, pedras, chão de floresta e agulhas de eucalipto. O processo de pintura também encontra-se na mesma aba de "*Landscape mode*" da seção acima, e envolve definir um "*Landscape layer info*", (Informações de camada de paisagem, arquivo responsável pelo armazenamento da informação de onde o material é aplicado) para cada uma das camadas presentes no *shader*. Aqui, é preciso tomar cuidado com onde cada material é aplicado: o *shader* é compilado para cada componente do terreno, sendo seu custo atrelado ao número de materiais presentes em cada um destes componentes. Por isso, torna-se interessante remover materiais que contribuem pouco para a texturização de determinadas fatias do terreno. Ademais, presta-se atenção nas áreas de transição entre um material e outro, objetivando evitar linhas definidas de separação.

Na Figura 39, a visualização da complexidade de *shader*, que mede o número de operações calculadas pela placa de vídeo, sendo que, neste caso, quanto mais vermelha a área, mais "cara" é sua renderização. Percebe-se diferença de cor entre certos componentes do terreno: verdes, dois materiais; marrons, três; terracotas, quatro.

[66] UNREAL SENSEI. **Unreal Engine 5 Landscape Master Material**. [202-]. Disponível em: https://www.unrealsensei.com/asset/ue5landscape. Acesso em: 18 nov. 2025.

Figura 39: Visualização da complexidade de cada seção do *landscape shader*



Fonte: Produzida pelo autor.

Além das camadas que possuem um único material, o *shader* também possui uma que aplica dois materiais distintos de maneira automática, baseando-se em quão íngreme o terreno é para definir os pontos de transição.

Para adicionar detalhe geométrico ao *landscape*, utiliza-se *Nanite* para tesselá-lo, como feito anteriormente nas cinzas das lareiras, por meio de textura de *displacement*. O *shader* desenvolvido permite definir quais camadas do terreno utilizarão tal técnica, que é mais perceptível em objetos com detalhes mais proeminentes. Neste caso, foi utilizado apenas nas pedras e no chão da floresta. Tal técnica tem elevado custo à performance (medições apontam para mais de 3ms por *frame*), mas resultado visual adequado às necessidades do projeto.

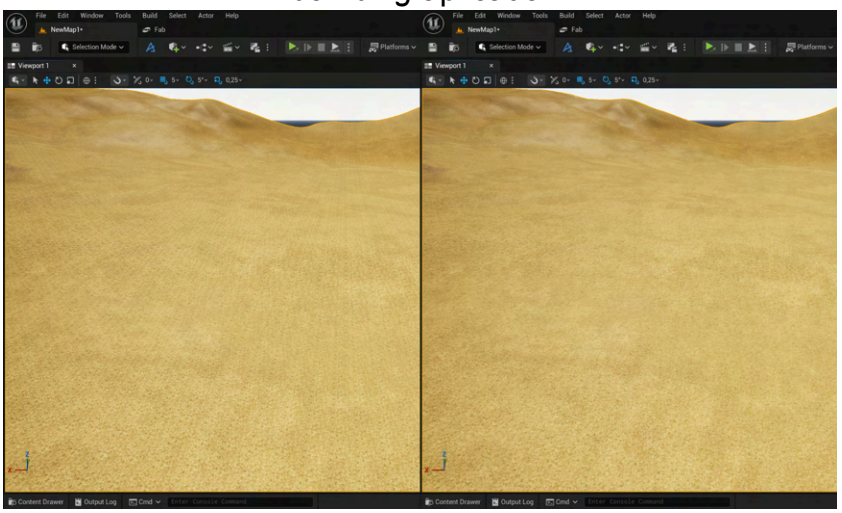
3.4.4.1.1 *Cell bombing*

Cell bombing[67] é uma técnica procedural que posiciona diversas imagens pequenas em intervalos irregulares a fim de reduzir artefatos visuais de repetição de textura. Neste caso, envolve utilizar uma textura de ruído Voronoi com intuito de randomizar as coordenadas UV, como visto na Figura 40. Uma imagem é posicionada em cada uma das células da textura Voronoi, randomizando sua

[67] GLANVILLE, R. Steven. Chapter 20. Texture Bombing. In: FERNANDO, Randima (org.). **GPU Gems**: programming techniques, tips and tricks for real-time graphics. [S.l]: Addison-Wesley Professional, 2004. Disponível em: https://developer.nvidia.com/gpugems/gpugems/part-iii-materials/chapter-20-texture-bombing. Acesso em: 18 nov. 2025.

posição, rotação e escala. Feito isso, usa-se de outra função para borrar os limites entre células. Em exemplo fornecido pela *Nvidia*, uma aplicação simples de *cell bombing*, em textura única, avalia esta mesma textura oito vezes para chegar no resultado desejado. É possível reduzir tal número “removendo a imagem final do *loop* interno, salvando as coordenadas da amostra resultante em uma variável e fazendo uma amostragem ao final do programa”[68] (tradução nossa). Tal truque não funciona caso seja preciso avaliar o canal alfa de uma imagem para saber se os valores são transparentes ou não.

Figura 40: Comparação entre *landscape shader* sem (esquerda) e com (direita) *cell bombing* aplicado.



Fonte: Produzida pelo autor.

3.4.4.1.2 *Channel packing*

Dentre os mais demorados processos em que a GPU trabalha, encontra-se *texture sampling*[69] (amostragem de textura), feita para cada imagem utilizada por um *shader* qualquer. Como forma de reduzir tal custo, materiais produzidos pela *Quixel* e disponíveis no *Fab* empacotam três mapas em uma única imagem, chamada de

[68] GLANVILLE, R. Steven. Chapter 20. Texture Bombing. In: FERNANDO, Randima (org.). **GPU Gems**: programming techniques, tips and tricks for real-time graphics. [S.l]: Addison-Wesley Professional, 2004. Disponível em: https://developer.nvidia.com/gpugems/gpugems/part-iii-materials/chapter-20-texture-bombing. Acesso em: 18 nov. 2025.

[69] GIESEN, Fabian. **A trip through the Graphics Pipeline 2011, part 4**. 2011. Disponível em: https://fgiesen.wordpress.com/2011/07/04/a-trip-through-the-graphics-pipeline-2011-part-4/. Acesso em: 18 nov. 2025.

ORM. Nela, o canal R é usado para o mapa de oclusão de ambientes (*ambient occlusion*); o mapa G é usado para a informação de rugosidade (*roughness*); e o mapa B é usado para metalicidade (*metalness*)[70]. Soma-se à esta os mapas de informação de cor e de normal, totalizando três texturas distintas para cada *shader*, podendo ainda aumentar para quatro caso seja necessária a presença de informação de *displacement*. Ao desenvolver o *shader* que seria utilizado para o terreno, houve necessidade de utilizar a técnica de *cell bombing* para reduzir a repetição aparente dos materiais. Como dito anteriormente, tal técnica é capaz de rapidamente multiplicar o número de passes de *texture sampling*, sendo então interessante empacotar ainda mais as texturas, de modo a reduzir o número de imagens utilizadas. Para isso, inicialmente, desenvolveram-se gráficos no aplicativo *Adobe Substance Designer* que transformam cinco mapas de textura em apenas dois:

- Imagem 1: canais R, G e B usados para a informação de cor;
    - Canal A usado para *roughness*;
    - Por motivo de organização, exportações de tal mapa foram seguidas do sufixo “CR”, denotando “*color*” e “*roughness*”.
- Imagem 2: canais R e G usados para o mapa de normal;
    - Canal B usado para *ambient occlusion*;
    - Canal A usado para o mapa de *displacement*;
    - Tal mapa foi seguido do sufixo “NOH”, denotando “*normal*”, “*occlusion*” e “*height*”.

O mapa de normal pode ser empacotado em apenas dois canais porque seu canal B pode ser reconstruído pelo motor gráfico em tempo de execução por meio do teorema pitagórico ($A^2+B^2=C^2$; neste caso, $R^2+G^2=B^2$). A *Unreal* possui uma *material function* chamada “*DeriveNormalZ*” que automaticamente calcula o canal B a partir dos outros dois canais.

Por via de comparação, um *shader* de *landscape* com quatro materiais usando o método de *channel packing* listado acima reduz o número de *texture samples* de 20 para 8.

[70] Funciona visto que os três mapas utilizados armazenam suas informações em escala de cinza.

Porém, durante o processo, percebeu-se que alguns problemas aparecem ao tentar empacotar um canal alfa junto de uma textura com informação de normal, visto que a Unreal automaticamente remove tal canal quando a opção de compressão da imagem é alterada para "*Normal*", para economizar a quantidade de memória alocada a textura. Por isso, decidiu-se utilizar três texturas distintas para cada camada do *shader*:

- Imagem 1: canais R, G e B usados para a informação de cor;
    - Canal A vazio.
- Imagem 2: canais R, G e B usados para a informação de normal;
    - Canal A vazio;
- Imagem 3: canal R usado para *ambient occlusion*;
    - Canal G usado para *roughness*;
    - Canal B usado para *displacement*;
    - canal A vazio.

Seria possível, então, adicionar ainda dois outros mapas a tais imagens, como *specular* e *cavity*, por exemplo, sem acarretar em custo à performance, visto que a imagem é avaliada para *texture sampling* como um todo, e não canal por canal.

3.4.4.1.3 *Virtual texture*

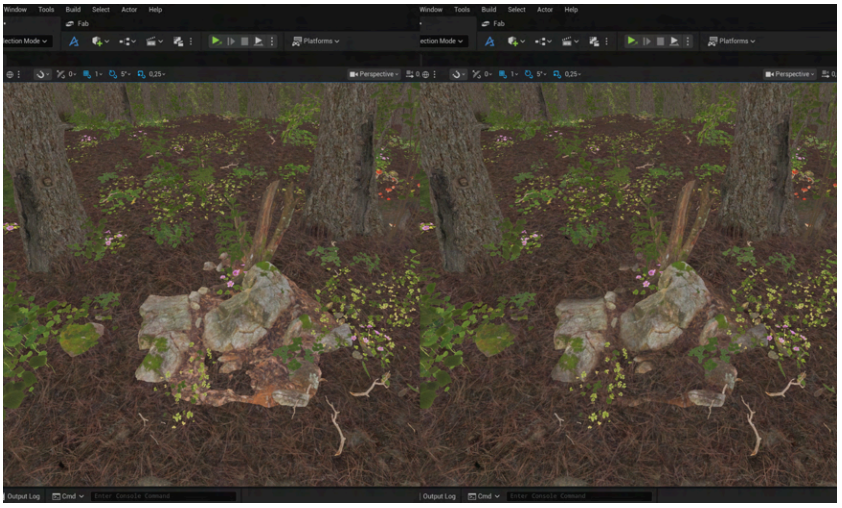
Virtual texture[71] é um método de manuseio de texturas de alta resolução. Usado neste projeto na modalidade "*Runtime virtual texture*" (Textura virtual em tempo de execução), gerada pela placa de vídeo e armazenada em memória. Baseia-se nas camadas de materiais usadas no *Landscape*, permitindo usá-la para mesclar as bases de malhas que não são automaticamente compatíveis com o *shader* do cenário.

---

[71] EPIC GAMES. **Runtime Virtual Texturing**: An overview of the runtime virtual texturing method. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/runtime-virtual-texturing-in-unreal-engine. Acesso em: 18 nov. 2025.

Para ativar sua funcionalidade, primeiro criam-se dois "*Runtime Virtual Texture Assets*", um para gravar a informação de cor da textura, e outro para gravar sua informação de altura. Neles, é possível configurar parâmetros relacionados à resolução de tal textura. Feito isso, cria-se então dois volumes que encompassam o *landscape* como um todo e atribui a cada um um *RVT asset* distinto. Adicionam-se então expressões fornecidas pela *Epic Games* aos materiais nos quais o efeito é desejado, de maneira a expor parâmetros nas instâncias de material que permitem controlar a forma como a interpolação entre o material base e a *RVT* acontece. Ao aplicar a textura virtual ao material, nota-se maior coesão na área de transição entre sua base e o terreno, como visto na Figura 41.

Figura 41: Comparação entre objeto sem (esquerda) e com (direita) textura virtual aplicada à sua base



Fonte: Produzida pelo autor.

3.4.5 Vegetação

Preferiu-se por utilizar *assets* novamente produzidos pela divisão de escaneamento 3D da *Epic Games*, *Quixel*. O processo envolve selecionar os objetos na biblioteca e adicioná-los à aba de "*Foliage mode*"[72]. Nela, é possível controlar a densidade de cada componente, a distância entre componentes idênticos, sua faixa

[72] EPIC GAMES. **Foliage Mode**: How to render Static Mesh or Actor Foliage on the surfaces of other geometry for creating ground cover effects. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/foliage-mode-in-unreal-engine. Acesso em: 18 nov. 2025.

de escala, se o objeto alinha-se com a direção das normais da malha em que está sendo pintado (útil para plantas pequenas, não tanto para árvores), limites para o ângulo da malha em que sua posição é considerada válida, se possui colisão; e diversas configurações de renderização, como se o objeto produz sombras, se tais sombras são dinâmicas (atualizam quadro a quadro), a maneira como tais mapas de sombras são descartados pelo motor gráfico para salvar em performance, se afeta a iluminação indireta, e se faz parte da estrutura de *ray tracing*.

Para funcionar em tal aba, é necessário que o ator seja do tipo "*Static mesh foliage*". A vegetação disponibilizada pelo *Quixel* já vem configurada dessa maneira, mas se esse não for o caso, é possível transformar uma "*Static mesh*" em uma "*Static mesh foliage*" diretamente na Unreal Engine (foi necessário fazer isto com os cogumelos presentes na cena, que não são importados da maneira correta, visto que *Quixel* não os considera vegetação). Isto se dá para redução de custo em performance: a vegetação gera um único ator, sendo que todas as malhas repetidas dentro da aba de pintura são tratadas como objetos instanciados, reduzindo sua pegada em memória.

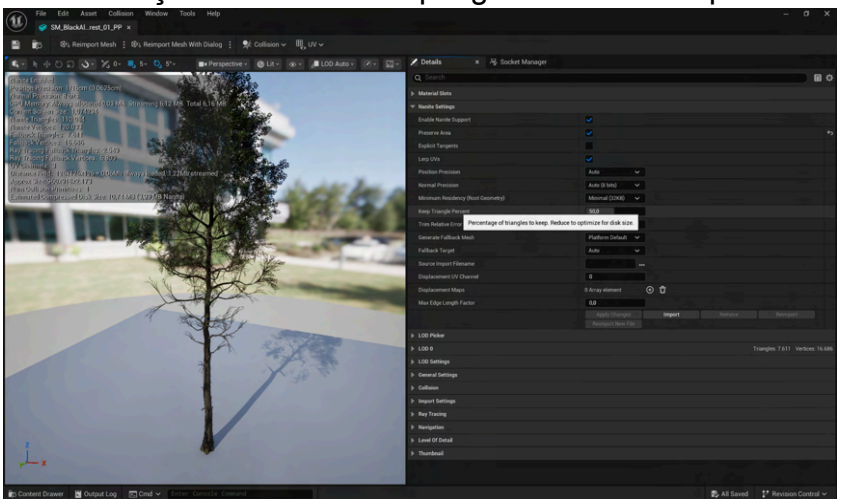
Após configuradas as malhas, o processo envolve escolher os objetos e pintá-los, sendo possível definir qual o tipo de ator (*Landscape*, *Static mesh*, *BSP*, *Foliage* ou *Translucent*) pode receber vegetação. Em alguns casos, objetos podem ser avaliados como inválidos (sua posição não condiz com os parâmetros que foram definidos), sendo necessário manualmente alterar suas posições, ou simplesmente deletá-los.

3.4.5.1 Considerações sobre vegetação

Ao baixar, as árvores são adicionadas ao projeto com *Nanite* ativado, sendo possível alterar o método de exibição para *LODs* tradicionais. Selecionando *Nanite*, controla-se a quantidade máxima de polígonos que serão renderizados em jogo por meio de um *slider* (Figura 42). Para reduzir o custo de renderização, optou-se por manter todas as árvores com valores máximos próximos a 100.000 polígonos. Com *LODs*, é possível impedir que níveis de maior detalhe sejam carregados, surtindo o mesmo efeito. Também alterou-se a maior resolução que as texturas de tais árvores

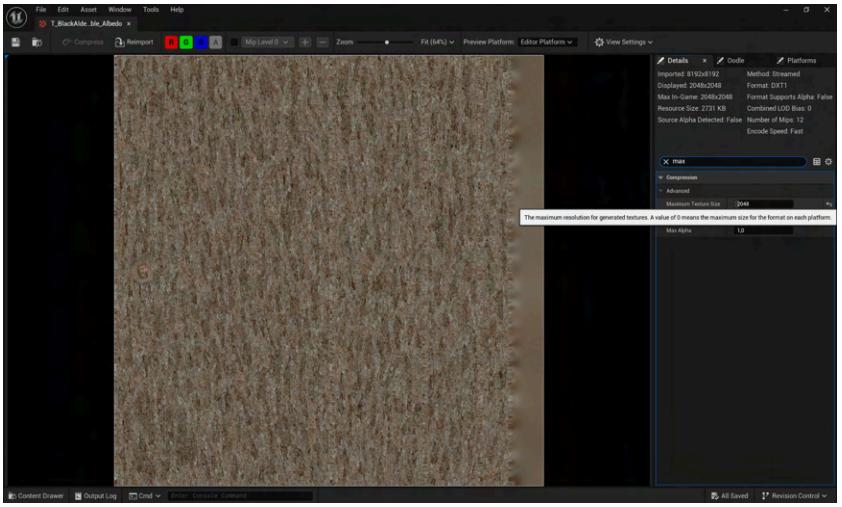
podem ser carregadas do padrão, que é 8K, para 2K, acarretando em redução de 93% do seu custo em memória (de 43.691 KB para 2.741 KB), como pode ser visto na Figura 43.

Figura 42: Visualização da *static mesh* de uma das árvores com a opção para redução do número de polígonos em destaque



Fonte: Produzida pelo autor.

Figura 43: Visualização de textura com a opção para reduzir seu tamanho máximo durante *gameplay* em destaque



Fonte: Produzida pelo autor.

Cada *static mesh* possui duas variações que diferenciam-se uma da outra apenas pela maneira como calculam vento: *Simple wind*, mais barato, possui um único ponto âncora em sua base, resultando em animações simplificadas; e *Pivot painter*, mais caro, usa de cores de vértice para definir gradientes de influência.

Desta maneira, áreas como a base do tronco e pontos de nascimento de galhos chacoalham menos do que a copa e as pontas dos galhos, resultando em animação mais convincente. Preferiu-se por usar as árvores *Pivot painter* neste projeto.

Anteriormente, alterou-se o método de geração de sombras da "*Directional light*" que representa o sol/lua de *ray tracing* para *VSMs*, visto os artefatos gerados em malhas com *Nanite*. Porém, levando em consideração a complexidade das formas, *VSMs* não funcionam bem com árvores.

Com *Nanite*, por causa dos problemas gerados com sombras *ray tracing*, é necessário usar *VSMs*. Testou-se também a possibilidade de utilizar *LODs* tradicionais. Porém, ao alterar o método de representação da geometria, o programa avisa que uma área muito grande está sendo avaliada para *VSMs*, resultando em degradação de performance, sendo então necessário utilizar *ray tracing*, que neste caso específico tem custo inferior à *VSMs*. Resumidamente, caso as árvores usem *Nanite*, a luz direcional deve gerar sombras por *VSMs*; e caso usem *LODs*, a luz direcional deve gerar sombras por *ray tracing*.

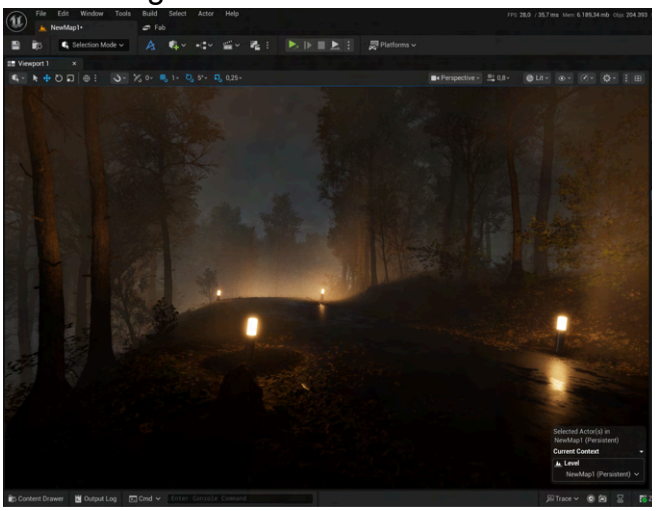
O projeto permitiu contornar tal impasse aumentando a cobertura de nuvens no céu, tornando-se possível desativar as sombras geradas pela luz direcional, já que não mais afeta o cenário. Desta maneira, a iluminação geral do nível depende unicamente daquela gerada pela *SkyLight*.

Caso exista a necessidade de que o sol/lua produzam sombras, é possível alterar o comportamento das sombras de cada objeto utilizado dentro da aba de vegetação. Por exemplo, é possível desativar sombras para malhas individuais, caso a contribuição à cena seja considerada diminuta (como é o caso de grama ou de objetos que acabam frequentemente com suas sombras cobertas por outros maiores). Para isso, desmarca-se a opção "*Cast shadow*". Ademais, é possível alterar a opção referente à avaliação de atualização dos mapas de sombra gerados pela luz direcional, retirando deles a necessidade de serem atualizados a cada quadro. Para isso, altera-se a opção "*Shadow cache invalidation behavior*" de "*Auto*" para "*Rigid*", que suprime atualizações de *WPO* (*World Position Offset*, deslocamento de posição global, usado para animação por *shader*, por exemplo, como é o caso das plantas produzidas pela *Quixel*); ou "*Static*", que adiciona ao "*Rigid*" invalidações devido a mudanças de transformação, como posição. Isto cria

uma desconexão entre o farfalhar das folhas, controlado pelo *Global foliage actor* citado acima, e sua sombra, sendo necessário avaliar o que é mais importante para o projeto: visual ou performance.

Anedoticamente, com atualização constante de sombras, cada quadro na floresta leva em torno de 36ms para ser renderizado; alterando para a opção "*Rigid*", cada quadro começa a ser renderizado em 24ms, uma redução de 33%. Abaixo, na Figura 44, o resultado final dos processos acima discutidos.

Figura 44: Floresta finalizada



Fonte: Produzida pelo autor.

3.4.6 *Cutscene* e *gameplay*

Após finalizado o cenário, voltam-se as atenções para a produção do arquivo audiovisual. Novamente usa-se de arquivos fornecidos pela *Epic*, fazendo alterações em *shaders* e *blueprints* à medida que mostraram-se necessárias.

3.4.6.1 Carro

Pouco tempo depois do lançamento da *Unreal Engine 5*, *Epic Games*, em conjunto com diversos estúdios, publica a demonstração "*The Matrix Awakens*"[73],

[73] EPIC GAMES. **Introducing The Matrix Awakens: An Unreal Engine 5 Experience**. 2021. Disponível em: https://www.unrealengine.com/en-US/blog/introducing-the-matrix-awakens-an-unreal-engine-5-experience. Acesso em: 18 nov. 2025.

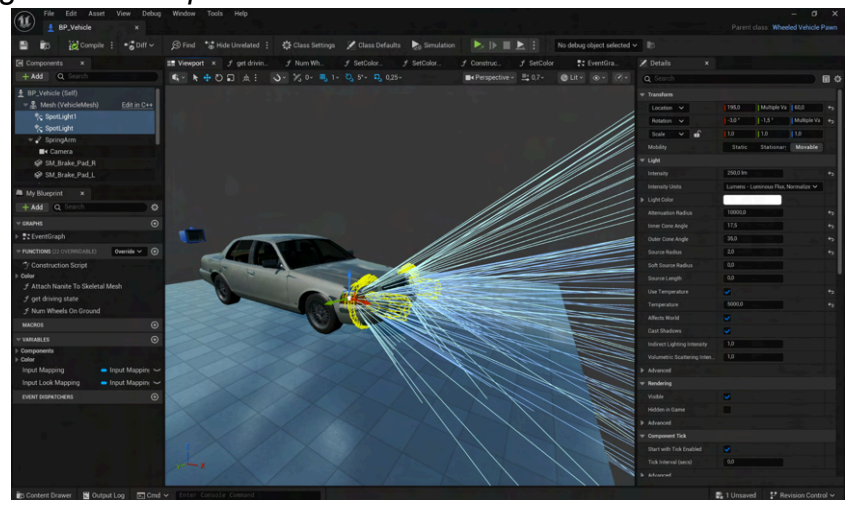
pondo em destaque as novas tecnologias desenvolvidas para a nova versão do motor gráfico. A demonstração em si fica disponível para *download* por apenas alguns meses, mas certos arquivos usados em seu desenvolvimento continuam à disposição dos usuários da *Unreal* até a data de produção deste relatório. Dentre eles, é fornecido um projeto que contém diversos modelos de automóveis.

Adicionou-se então uma das variantes presentes em tal arquivo ao nível, já com *rig*, colisão, controles e grande parte dos *shaders* configurados. Foram necessárias apenas algumas alterações para adequá-lo às necessidades da *cutscene*.

Primeiramente, percebeu-se que o modelo não possuía luzes funcionais. Configurou-se então os vidros das luzes para possuírem propriedade emissiva, não gerando resultado adequado. Após vasculhar os arquivos baixados, encontrou-se uma textura que identifica áreas emissivas do veículo, sendo necessário adicioná-la ao *shader* das lanternas por meio de sucessivas multiplicações e adições. Expondo as variáveis à instância de material, torna-se possível controlar o comportamento das luzes dianteiras, luzes traseiras, luz de ré e luzes de emergência. Porém, considerando a maneira como a textura foi produzida, não foi possível desatrelar o acendimento das luzes dianteiras da luz de ré.

Ademais, também foi necessário adicionar duas *spotlights* à *blueprint* geral dos automóveis, como visto na Figura 45. Primeiramente, tentou-se fazê-lo diretamente na *viewport*, mas, ao adicionar o modelo ao *Sequencer*, as luzes permaneciam estáticas enquanto o carro se movia.

Figura 45: *Blueprint* dos veículos com os holofotes selecionados

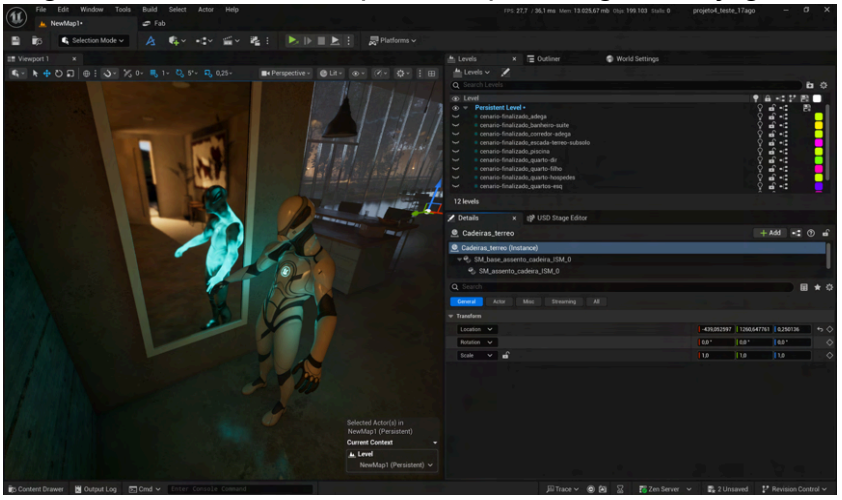


Fonte: Produzida pelo autor.

3.4.6.2 Personagem

Quando o projeto foi criado, optou-se por usar o *template* de jogo em primeira pessoa disponibilizado pela *Epic Games*. Assim, vários *assets* úteis, como por exemplo um personagem controlável, são automaticamente adicionados ao arquivo. Porém, ao começar o planejamento para a gravação do *gameplay*, percebeu-se que a malha utilizada na *blueprint* controlada seria visível em reflexos. Ademais, ao utilizar *Surface cache* como método de iluminação global, a simplificação gerada para o modelo é dominada por *shader* emissivo de cor turquesa (Figura 46).

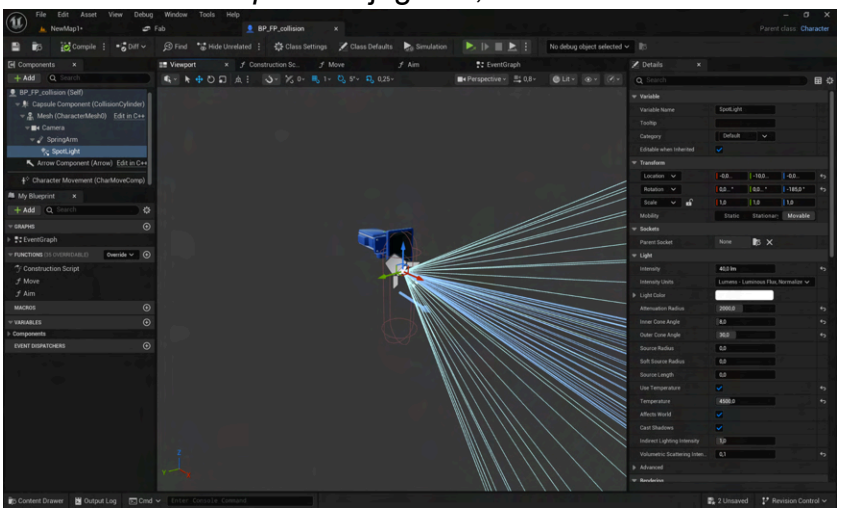
Figura 46: Malha usada pela *blueprint* original do jogador



Fonte: Produzida pelo autor.

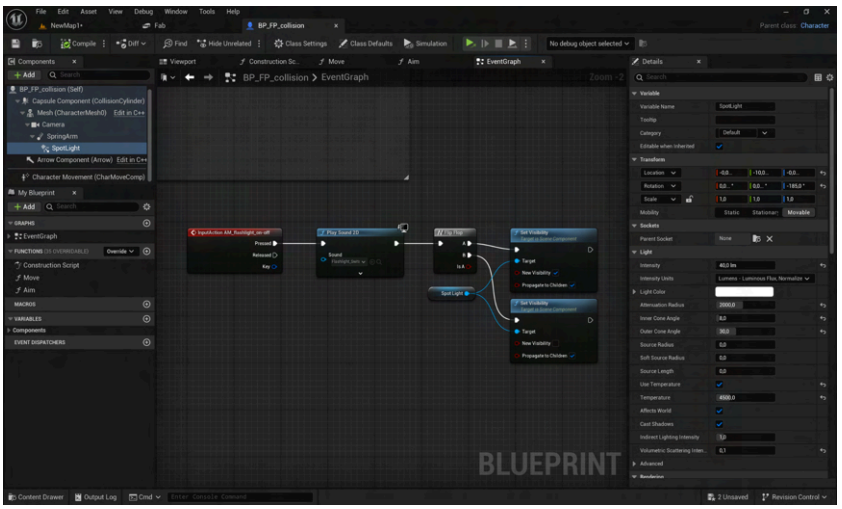
Considerando isto, desenvolveu-se nova blueprint, sem malha atrelada (Figura 47). Adicionou-se à ela também uma lanterna funcional, usando a *light function* citada anteriormente, que pode ser ligada e desligada com o pressionar de um botão (Figura 48).

Figura 47: Nova *blueprint* do jogador, com a lanterna selecionada



Fonte: Produzida pelo autor.

Figura 48: Nódulos adicionados ao *EventGraph* da *blueprint* para controlar o funcionamento da lanterna



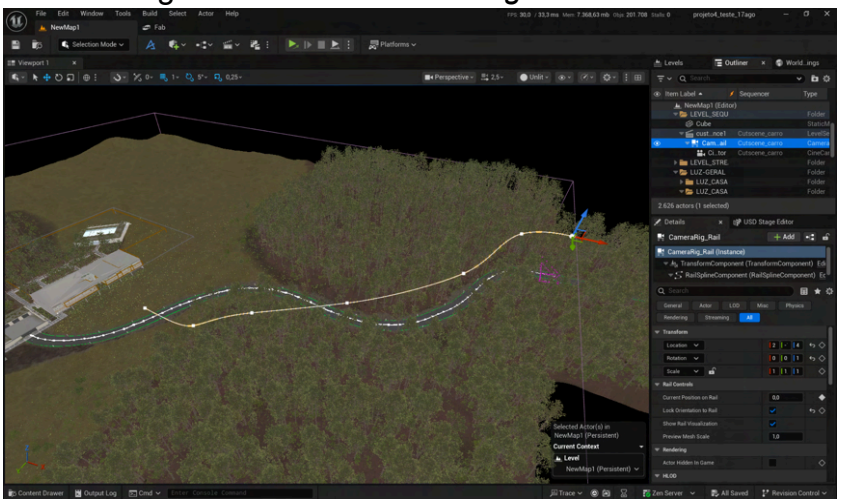
Fonte: Produzida pelo autor.

3.4.6.3 Gravação da *cutscene*

Primeiramente, adicionou-se um ator "*Camera Rig Rail*" à cena, que gera uma curva bézier (Figura 49). Com tal ator posicionado, é possível parentear uma câmera

a ele, adicionando *keyframes* que controlam a posição relativa entre a câmera e a curva.

Figura 49: Ator *Camera Rig Rail* na cena



Fonte: Produzida pelo autor.

Para a animação do carro, preferiu-se por controlá-lo em *gameplay* e usar a função *Take recorder*[74] para gravar as posições ocupadas pelo seu *rig*. Adicionou-se então tal tomada gerada à sequência que contém a animação da câmera. Por sorte, houve sincronização entre as duas já em primeiro momento, não necessitando animar o carro mais de uma vez.

4 CONSIDERAÇÕES FINAIS

Considera-se como alcançado o objetivo de analisar as etapas referentes à produção de design de níveis e arte de ambientes para jogos 3D. Permitiu-se discutir processos de maneira holística, iniciando com elaboração narrativa, partindo para coleta de referências e subsequente produção tanto em *software* de modelagem e texturização, quanto em motor gráfico. Percebe-se qualidade adequada de renderização, utilizando técnicas modernas como *ray tracing* e geometria virtualizada.

---

[74] EPIC GAMES. **Take Recorder**: Record Editor, Gameplay, and Live Link Actors with Take Recorder. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/take-recorder-in-unreal-engine. Acesso em: 18 nov. 2025.

Ao analisar a complexidade pretendida nas etapas de concepção do projeto, foi fundamental ao seu bom funcionamento iniciar a etapa de produção ainda no primeiro semestre de 2025, totalizando assim nove meses de trabalho. Fazendo isso, grande parte da modelagem arquitetônica e de *props* encontrou-se finalizada ao iniciar o segundo semestre de 2025, permitindo desta maneira separar mais tempo para o estudo de *software*, a *Unreal Engine 5*.

Sobre a *Unreal Engine*, foi utilizada durante o projeto em sua versão de número 5.6. Antes da entrega, a atualização 5.7 foi lançada, acarretando mudança de paradigma na renderização de vegetação em tempo real, como visto na demonstração *The Witcher 4 - Unreal Engine 5 Tech Demo*[75], por meio da criação de novo método de customização de árvores e afins, chamado *Procedural Vegetation Editor*[76] (*PVE*). Caso o projeto tivesse início em data posterior a sua finalização, usaria-se de tais técnicas para arborizar o cenário.

O restante da graduação também mostrou-se indispensável. Conhecimentos adquiridos em semestres anteriores, como uso de cores, composição, *mood*, animação, modelagem e iluminação foram amplamente usados em diferentes pontos do processo criativo, trabalhando em conjunto com os conhecimentos adquiridos durante os últimos dois semestres para elevar o projeto como um todo. Ademais, como dito anteriormente, arte para ambientes, como diversas outras áreas artísticas, é uma etapa altamente colaborativa. O auxílio de amigos, colegas e professores, seja ele com dicas, incentivos e *feedbacks*, ou com a modelagem e texturização de objetos, prova-se insubstituível. Sua exclusão fundamentalmente empobreceria aquilo que foi produzido.

Ao mesmo tempo, é impossível negar que várias dificuldades apresentaram-se durante a execução. Chama-se atenção para o processo de solução dos problemas relacionados à instâncias, que demandou mais de uma semana de retrabalho para adequar o arquivo exportado do *Blender* às necessidades do projeto na *Unreal Engine*. Porém, concomitantemente,

---

[75] **The Road to 60 FPS in The Witcher 4 Unreal Engine 5 Tech Demo | Unreal Fest Orlando 2025**. Orlando, FL, EUA: Unreal Engine, 2025. (47 min.), son., color. Disponível em: https://www.youtube.com/watch?v=ji0Hfiswcjo. Acesso em: 18 nov. 2025.

[76] EPIC GAMES. **Procedural Vegetation Editor (PVE)**: Use the Procedural Vegetation Editor to generate realistic, high-quality, Nanite-ready vegetation. 2025. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-vegetation-editor-pve-in-unreal-engine. Acesso em: 18 nov. 2025.

aprendeu-se muito sobre o funcionamento de processos relacionados à elaboração de espaços jogáveis, o que fazer e certamente o que não fazer.

Objetivando adequar-se aos prazos estipulados, partes do nível não passaram pelo mesmo detalhamento, encontrando-se mais próximas do estado de *blockout*. Tal processo de compartimentação permitiu alocar esforços para áreas mais fundamentais ao recorte escolhido. Ademais, a elaboração de certas mecânicas, como a abertura de portas, que permitiriam maior interação com o espaço foram consideradas fugindo um pouco das aspirações do projeto, sendo relegadas a data posterior.

Espera-se que tal relatório sirva como base para todos aqueles que busquem aprofundar-se em arte para ambientes, usando o que foi discutido nestas páginas para a elaboração de técnicas e espaços que adequem-se às suas aspirações.

REFERÊNCIAS

BARCLAY, Michael. **The Last of Us Part II, Downtown Seattle**. 2021. Disponível em: https://mikebarclay.co.uk/blocktober-2020/. Acesso em: 16 nov. 2025.

CLARKE, Susanna. **Piranesi**. Nova Iorque: Bloomsbury Publishing, 2020. 272 p.

DESAMSETTI, Harshith; LAL, Karu. Being a Realistic Master: Creating Props and Environments Design for AAA Games. **Asian Journal Of Humanity, Art And Literature**, [S.L.], v. 6, n. 2, p. 193-202, 31 dez. 2019. ABC Journals. http://dx.doi.org/10.18034/ajhal.v6i2.701.

DOUGBOMB. **Blue Prince**. [S.l]: Raw Fury, 2025. Jogo eletrônico.

EPIC GAMES. **Adding Detail Textures**: Guide for adding Detail Textures to your Materials. [201-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/adding-detail-textures?application_version=4.27. Acesso em: 18 nov. 2025.

EPIC GAMES. **Creating and Using LODs**: How To Create and Use LODs. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-and-using-lods-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Creating Landscapes**: Guide to creating new Landscape terrains. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/creating-landscapes-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Cubemaps**: The landing page for cubemaps: creation, export, import, and usage within Unreal Engine 4. [201-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/cubemaps?application_version=4.27. Acesso em: 18 nov. 2025.

EPIC GAMES. **Decal Materials**: An overview of decal materials and how to use them in your projects. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/decal-materials-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Exponential Height Fog**: An overview of the height-based, distant fog system. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/exponential-height-fog-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Foliage Mode**: How to render Static Mesh or Actor Foliage on the surfaces of other geometry for creating ground cover effects. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/foliage-mode-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Hardware Ray Tracing**: An overview of the features that use hardware to render real-time ray traced results. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-ray-tracing-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Introducing The Matrix Awakens: An Unreal Engine 5 Experience**. 2021. Disponível em: https://www.unrealengine.com/en-US/blog/introducing-the-matrix-awakens-an-unreal-engine-5-experience. Acesso em: 18 nov. 2025.

EPIC GAMES. **Level Streaming Overview**: Streaming Levels can be loaded with Level Streaming Volumes or programmatically with Blueprints or C++. [20--]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/level-streaming-overview-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Light Functions**: An overview of using materials to mask a light's projected area. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-light-functions-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Lumen Technical Details**: Dive into the technical details of using Lumen's global illumination and reflections features with software or hardware ray tracing. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-technical-details-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **MegaLights**: A whole new direct lighting path to place many dynamic and shadowed area lights in a scene. 2025. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/megalights-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Mesh Distance Fields**: An overview of Mesh Distance Fields and its available features that you can use when developing your games. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/mesh-distance-fields-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Mesh Paint Mode Tools and Settings**: Descriptions and usages for the various paint tools and settings of the Mesh Paint Mode. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/mesh-paint-tool-reference-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Nanite Virtualized Geometry Overview**: Learn about Nanite's virtualized geometry system and how it achieves pixel scale detail and high object counts. [202-]. Disponível em:

https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Procedural Vegetation Editor (PVE)**: Use the Procedural Vegetation Editor to generate realistic, high-quality, Nanite-ready vegetation. 2025. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/procedural-vegetation-editor-pve-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Ray Tracing Performance Guide**: A selection of topics for improving performance of ray tracing features in your projects. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/ray-tracing-performance-guide-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Runtime Virtual Texturing**: An overview of the runtime virtual texturing method. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/runtime-virtual-texturing-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Take Recorder**: Record Editor, Gameplay, and Live Link Actors with Take Recorder. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/take-recorder-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Virtual Shadow Maps**: Learn about the high-resolution shadow techniques designed with film-quality assets and large dynamically lit open worlds in mind. [202-]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/virtual-shadow-maps-in-unreal-engine. Acesso em: 18 nov. 2025.

EPIC GAMES. **Visibility and Occlusion Culling**: An overview of available visibility and occlusion culling methods. [20--]. Disponível em: https://dev.epicgames.com/documentation/en-us/unreal-engine/visibility-and-occlusion-culling-in-unreal-engine. Acesso em: 18 nov. 2025.

EXTERMÍNIO. Direção de Danny Boyle. Roteiro: Alex Garland. Londres: DNA Films, 2002. (113 min.), son., color.

EX MACHINA. Direção e roteiro de Alex Garland. Reino Unido: Film4, 2014. (108 min.), son., color.

Fab. Disponível em: https://fab.com. Acesso em: 18 nov. 2025.

GERASCH, Lucian. **USDHook4Collisions**. Disponível em: https://github.com/LucianGerasch/USDHook4Collisions. Acesso em: 19 nov. 2025.

GIANT SPARROW. **What Remains of Edith Finch**. [S. l.]: Annapurna Interactive, 2017. Jogo eletrônico.

GIESEN, Fabian. **A trip through the Graphics Pipeline 2011, part 4**. 2011. Disponível em: https://fgiesen.wordpress.com/2011/07/04/a-trip-through-the-graphics-pipeline-2011-part-4/. Acesso em: 18 nov. 2025.

GIL VIDAL, Vinícius. **Modelagem 3D de Personagens e Objetos para Jogos**. 2025. 69 f. TCC (Graduação) - Curso de Animação, Centro de Comunicação e Expressão, Universidade Federal de Santa Catarina, Florianópolis, 2025. Disponível em: https://repositorio.ufsc.br/handle/123456789/267308. Acesso em: 20 nov. 2025.

GLANVILLE, R. Steven. Chapter 20. Texture Bombing. In: FERNANDO, Randima (org.). **GPU Gems**: programming techniques, tips and tricks for real-time graphics. [S.I.]: Addison-Wesley Professional, 2004. Disponível em: https://developer.nvidia.com/gpugems/gpugems/part-iii-materials/chapter-20-texture-bombing. Acesso em: 18 nov. 2025.

IO INTERACTIVE. **Hitman World of Assassination**. Copenhague, DK; Warner Bros., 2022. Jogo eletrônico.

KARLSSON, Tobias; BRUSK, Jenny; ENGSTRÖM, Henrik. **Level Design Processes and Challenges**: a cross section of game development. Games And Culture, [S.L.], v. 18, n. 6, p. 821-849, 20 nov. 2022. SAGE Publications. http://dx.doi.org/10.1177/15554120221139229.

KHRONOS GROUP. **glTF - what the duck?**: An overview of the basics of the GL Transmission Format. [20--]. Disponível em: https://www.khronos.org/files/gltf20-reference-guide.pdf. Acesso em: 18 nov. 2025.

LOUX, Xavier. **Blender for Unreal Engine**. Disponível em: https://github.com/xavier150/Blender-For-UnrealEngine-Addons. Acesso em: 19 nov. 2025.

MATT NEWELL. **Lushfoil Photography Sim**. [S.I.]; Annapurna Interactive, 2025. Jogo eletrônico.

MCMILLAN, Luke. **A Rational Approach To Racing Game Track Design**. In this extensive article, Gamasutra takes an in-depth look at racing game track design, comparing two arcade titles -- *Initial D* and *Maximum Tune* -- and contrasting them, at important points, against the approach used in the *Gran Turismo* series.2011. Disponível em: https://www.gamedeveloper.com/design/a-rational-approach-to-racing-game-track-design. Acesso em: 20 nov. 2025.

MILLER, Jay. **Call of Duty: Modern Warfare - SP - Clean House**. 2020. Disponível em: https://www.artstation.com/artwork/aYNQXX. Acesso em: 18 nov. 2025.

NAUGHTY DOG. **The Last of Us Part I**. Santa Mônica, CA, EUA; Sony Interactive Entertainment, 2022. Jogo eletrônico.

NAUGHTY DOG. **The Last of Us Part II**. Santa Mônica, CA, EUA; Sony Interactive Entertainment, 2020. Jogo eletrônico.

NAUGHTY DOG. **Uncharted 4: A Thief's End**. Santa Mônica, CA, EUA; Sony Interactive Entertainment, 2016. Jogo eletrônico.

NEWZOO. **Global Games Market Report**. [S.l.]: Newzoo, 2025. 78 p. Disponível em: https://resources.newzoo.com/hubfs/Free%20Reports/Games%20Market%20Report%20and%20Forecasts/2025_Newzoo_Free_Global_Games_Market_Report.pdf Acesso em: 20 nov. 2025.

NOSFERATU, o Vampiro. Direção de F. W. Murnau. Roteiro: Henrik Galeen. República de Weimar: Prana Film, 1922. (94 min.), P&B.

NVIDIA. **Life of a triangle - NVIDIA's logical pipeline**. [20--]. Disponível em: https://developer.nvidia.com/content/life-triangle-nvidias-logical-pipeline. Acesso em: 18 nov. 2025.

PARASITA. Direção de Bong Joon Ho. Roteiro de Bong Joon Ho e Han Jin-won. Coreia do Sul: Barunson E&A, 2019. (132 min.), son., color.

Polligon. Disponível em: https://polligon.com. Acesso em: 18 nov. 2025.

Poly Haven. Disponível em: https://polyhaven.com. Acesso em: 18 nov. 2025.

REMEDY ENTERTAINMENT. **Alan Wake II**. Espoo, FI: Epic Games Publishing, 2023. Jogo eletrônico.

REMEDY ENTERTAINMENT. **Control**. Espoo, FI: 505 Games, 2019. Jogo eletrônico.

ROCKSTAR GAMES. **Red Dead Redemption II**. [S.l.]; Rockstar Games, 2018. Jogo Eletrônico.

textures.com. Disponível em: https://textures.com. Acesso em: 18 nov. 2025.

THE FULLBRIGHT COMPANY. **Gone Home**. [S. l.]: The Fullbright Company, 2013. Jogo eletrônico.

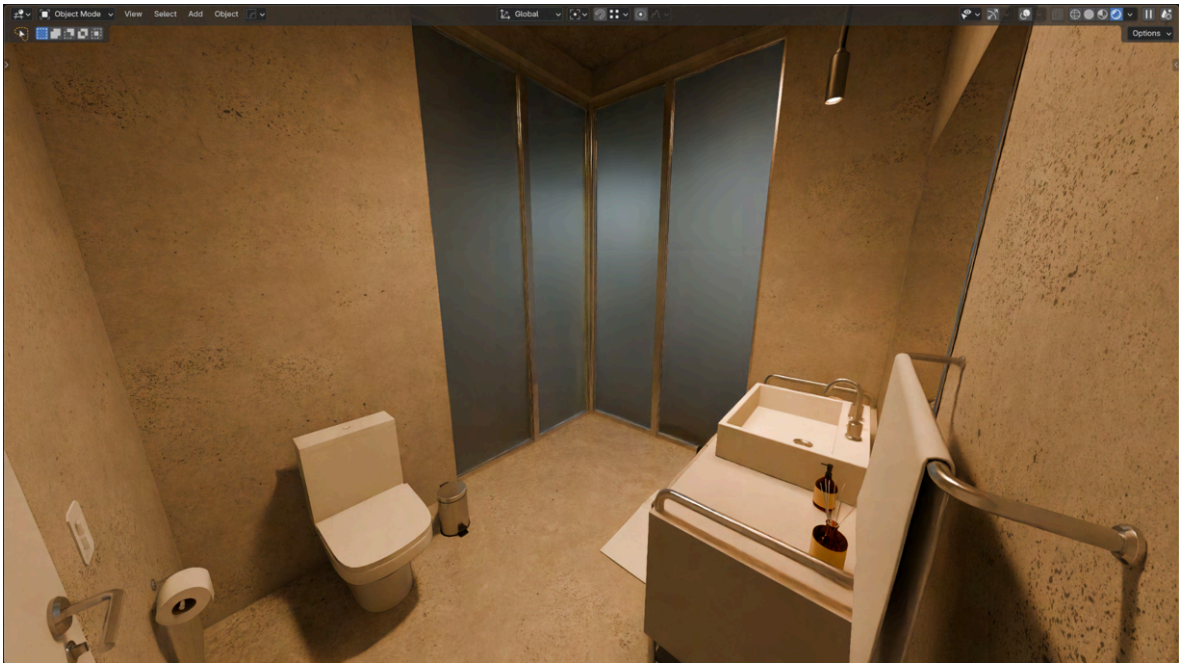
The Road to 60 FPS in The Witcher 4 Unreal Engine 5 Tech Demo | Unreal Fest Orlando 2025. Orlando, FL, EUA: Unreal Engine, 2025. (47 min.), son., color. Disponível em: https://www.youtube.com/watch?v=ji0Hfiswcjo. Acesso em: 18 nov. 2025.

TROIKA GAMES. **Vampire: The Masquerade - Bloodlines**. [S. l.]: Activision, 2004. Jogo eletrônico.

UNREAL SENSEI. **Unreal Engine 5 Landscape Master Material**. [202-]. Disponível em: https://www.unrealsensei.com/asset/ue5landscape. Acesso em: 18 nov. 2025.

APÊNDICE A – RESULTADOS NO *BLENDER*

Modelagem do lavabo



Fonte: Produzida pelo autor.

Modelagem do banheiro do térreo

Fonte: Produzida pelo autor.

Modelagem do quarto de hóspedes

Fonte: Produzida pelo autor.

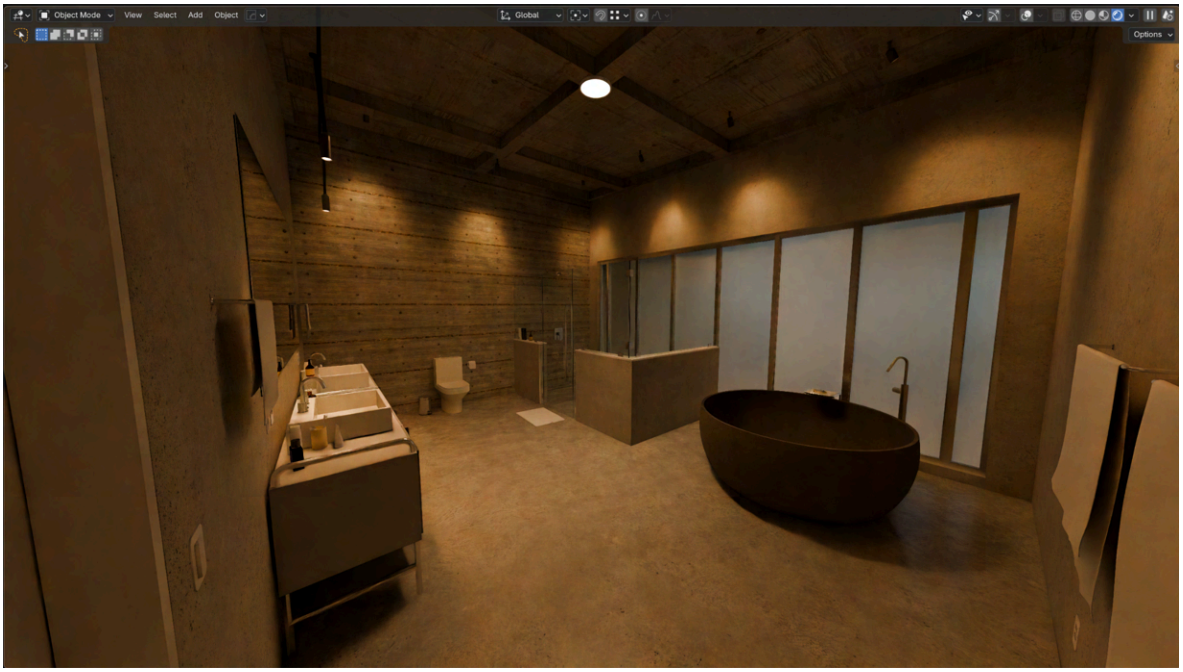
Modelagem do banheiro do quarto de hóspedes

Fonte: Produzida pelo autor.

Modelagem do *closet* da suíte master

Fonte: Produzida pelo autor.

Modelagem do banheiro da suíte master



Fonte: Produzida pelo autor.

Modelagem do quarto do filho

Fonte: Produzida pelo autor.

Modelagem do banheiro do filho

Fonte: Produzida pelo autor.

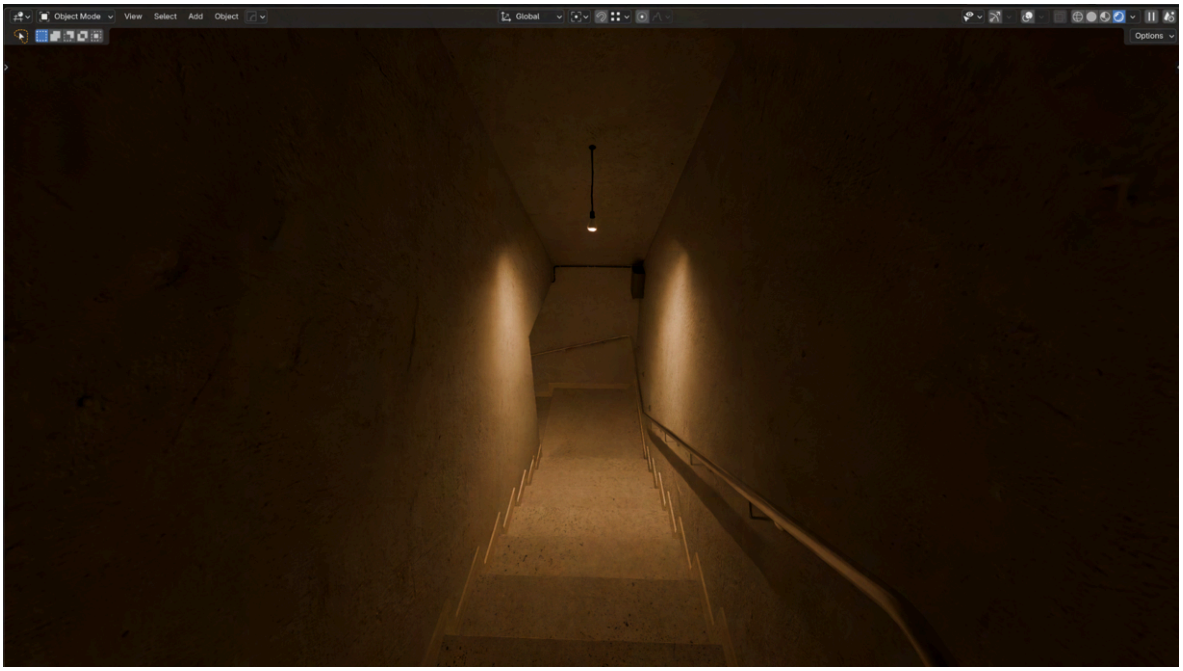
Modelagem da piscina

Fonte: Produzida pelo autor.

Modelagem da cozinha da piscina

Fonte: Produzida pelo autor.

Modelagem da primeira escada de acesso ao subsolo

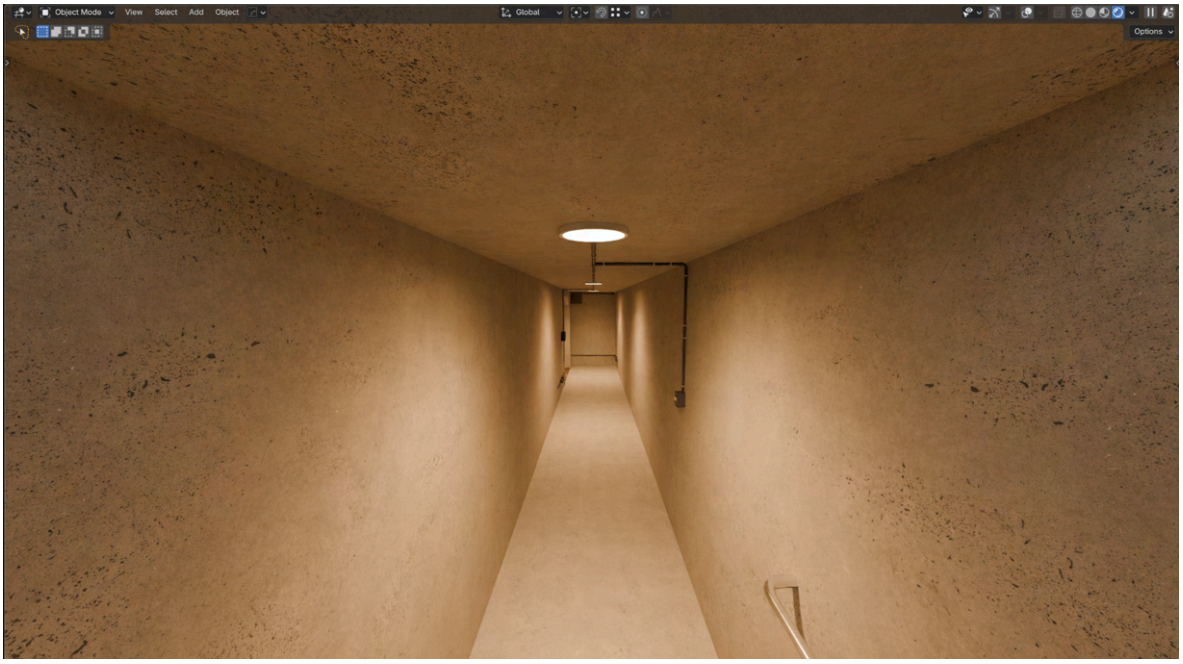


Fonte: Produzida pelo autor.

Modelagem do quarto de lenha

Fonte: Produzida pelo autor.

Modelagem do acesso de funcionários ao subsolo



Fonte: Produzida pelo autor.

Modelagem da porta de entrada do subsolo

Fonte: Produzida pelo autor.

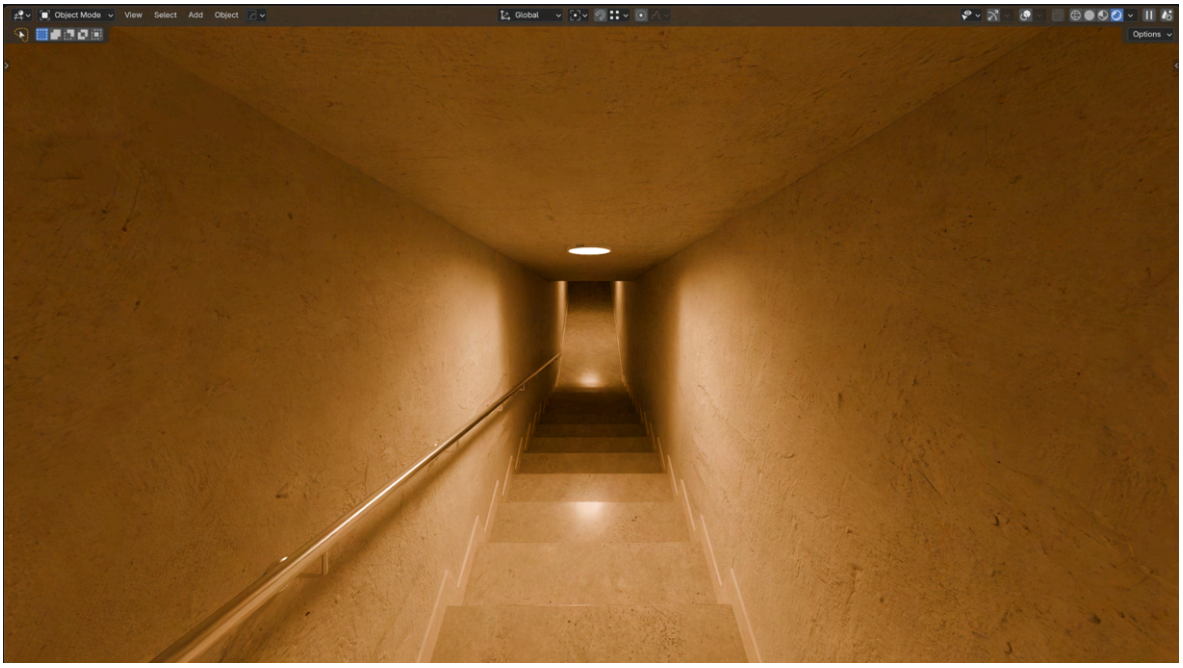
Modelagem do corredor geral do subsolo

Fonte: Produzida pelo autor.

Modelagem da área para lixo

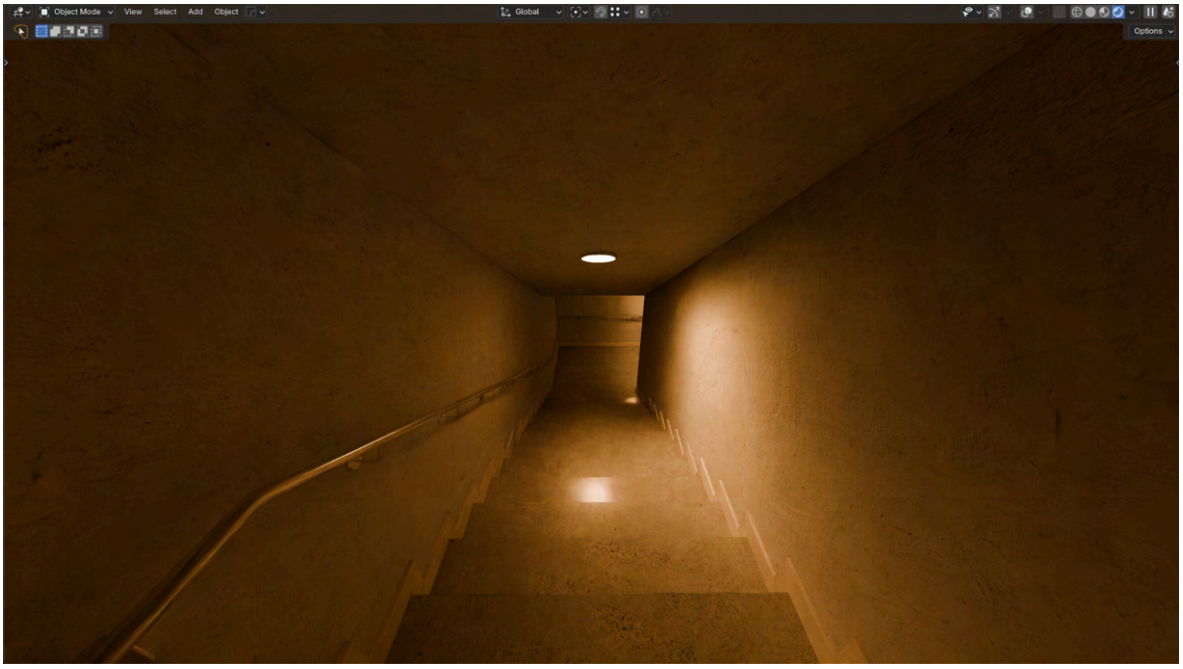
Fonte: Produzida pelo autor.

Modelagem da escada entre o subsolo e a adega



Fonte: Produzida pelo autor.

Modelagem da escada entre o escritório e a adega

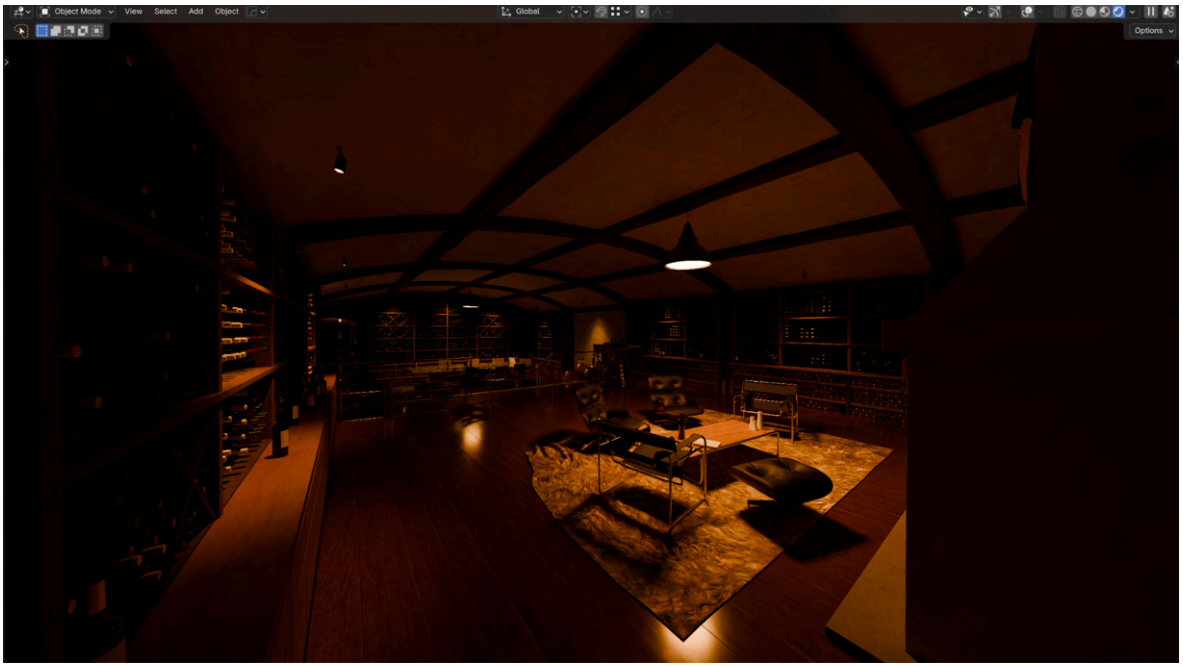


Fonte: Produzida pelo autor.

Modelagem da entrada da adega

Fonte: Produzida pelo autor.

Modelagem da adega



Fonte: Produzida pelo autor.

Modelagem da lareira

Fonte: Produzida pelo autor.

Modelagem da mesa

Fonte: Produzida pelo autor.

APÊNDICE B - RESULTADOS NA *UNREAL ENGINE*

Lixo na garagem

Fonte: Produzida pelo autor.

Corredor entre casa e garagem visto de frente

Fonte: Produzida pelo autor.

Corredor entre casa e garagem visto de trás

Fonte: Produzida pelo autor.

Sala de estar do primeiro andar

Fonte: Produzida pelo autor.

Sala de estar do térreo

Fonte: Produzida pelo autor.

Parte de trás da casa vista da sacada da suíte master

Fonte: Produzida pelo autor.

Parte de trás da casa

Fonte: Produzida pelo autor.

Vista alternativa da parte de trás da casa

Fonte: Produzida pelo autor.