UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

Graciliano Garcia Torres Galindo Jr

Um conjunto de métodos de otimização e boas práticas para jogos digitais 3D na portabilidade para plataformas menos potentes

Recife

Dezembro de 2018

2018

Graciliano Garcia Torres Galindo Jr

Um conjunto de métodos de otimização e boas práticas para jogos digitais 3D na portabilidade para plataformas menos potentes

Trabalho de Graduação

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Engenharia da Computação, orientada pelo professor Geber Lisboa Ramalho

Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

Recife

Dezembro de 2018

Graciliano Garcia Torres Galindo Jr

Um conjunto de métodos de otimização e boas práticas para jogos digitais 3D na portabilidade para plataformas menos potentes

Trabalho de Graduação

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Engenharia da Computação

Trabalho aprovado. Recife, 11 de Dezembro de 2018:

_____

Prof. Dr. Geber Lisboa Ramalho
Orientador

_____

Profa. Dra. Veronica Teichrieb
Avaliadora

Recife
Dezembro de 2018

Dedico este trabalho aos meus pais e irmã, que sempre sempre demonstraram total apoio na minha caminhada profissional.

Agradecimentos

Gostaria de agradecer ao meu orientador Geber Lisboa Ramalho pelo apoio e suporte na construção deste trabalho, bem como seus conselhos que foram essenciais para o mesmo.

Queria agradecer também a todos os professores que foram importantes para minha formação, a todos os amigos e colegas de curso que me apoiaram e me fizeram seguir em frente, a todos os familiares e as pessoas mais importantes na minha vida que me ajudaram e me deram a mão nesta nesta caminhada.

Em especial gostaria de agradecer a Felipe Nunes Walmsley, João Paulo Siqueira Lins e Thyeri Bione Marques, que estiveram presentes e disponíveis a me ajudar sempre que possível, durante o curso e na produção deste trabalho.

Resumo

Durante o desenvolvimento de jogos digitais 3D, muitas vezes são necessárias otimizações gráficas e de processamento, para obter uma experiência confortável. Porém, quando se está em estágio avançado ou finalizado do desenvolvimento e se decide fazer a portabilidade destes jogos para plataformas menos potentes como consoles, smartphones e similares, várias das otimizações já foram feitas, mostrando nesta situação a dificuldade em melhorar ainda mais a performance. Por este motivo, outros tipos de otimização e diminuição de qualidade são necessários. Este trabalho sugere um conjunto de métodos de otimização extras, junto a técnicas para mitigar e compensar a visibilidade dos efeitos negativos na qualidade gráfica e na maneira que o ambiente e os objetos são renderizados. Estes métodos são avaliados em estudo de caso de jogos já lançados em situação relevante. Através de um experimento e pesquisa de opinião sobre o método considerado mais importante, se vê que os métodos são, não apenas necessários, mas de melhor qualidade visual.

Palavras-chave: Renderização, Otimização, 3D, Qualidade Visual, Performance, Portabilidade

Abstract

In the middle of 3D game development, a lot of times there's the need of graphical and processing optimizations, in order to obtain a pleasant experience. Although, when in an advanced stage of development, or even when it's already released, and there's the decision for doing the porting of a game to other platforms, which are less powerful in terms of computing, like gaming consoles, smartphones and others, at that time most of the optimizations have already been done, so it shows as an issue to get the performance even better. For that reason, other types of optimization and decrease of quality are well needed. This work proposes a set of extra optimization methods, together with techniques to mitigate and compensate the visibility of the negative effects on the visual quality and on the way that the environment and the objects are rendered. These methods are evaluated in a case study with released games which remark this situation. There was an experiment and a survey about the method considered most important, and we conclude that the methods are not only necessary, but they also improve the visual quality of the experiences.

Keywords: Rendering, Optimization, 3D, Visual Quality, Performance, Porting

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

Sigla	Significado
LOD	*Level of Detail,* ou Nível de Detalhe
UE4	*Unreal Engine 4*
GPU	*Graphics Processing Unit,* ou *Unidade de Processamento Gráfico*
HLSL	*High Level Shading Language*
DoF	*Depth of Field*

1. Introdução

1.1. Contexto

No escopo da produção de jogos digitais 3D, todo o processo de desenvolvimento tem se tornado cada vez mais complexo, graças à demanda por melhor qualidade dos produtos. Fatores como performance fluida, alta fidelidade gráfica e menores tempos de carregamento são esperados dos jogos modernos no que se refere às escolhas de técnicas a serem utilizadas no desenvolvimento, ainda mais com a entrada de novos ambientes mais custosos, como o de Realidade Virtual.

Com tais necessidades, é comum trilhar o desenvolvimento de um produto com foco na(s) plataforma(s) escolhida(s). Quando sabe-se que um jogo em produção será lançado para uma certa plataforma, as escolhas de projeto serão feitas pensando nela, e os testes de execução serão feitos em unidades específicas. Isso impacta não só na performance, como no visual, pois a capacidade de processamento e reprodução gráfica da plataforma vão ditar o quão limitadas são as decisões de projeto[16].

Um dos exemplos de aspectos visuais que são mais afetados nestas decisões é o *Mesh Culling*, ou a decisão sobre renderização ou não de objetos em uma cena. Em uma plataforma com mais capacidade de processamento, muitos objetos poderão ser vistos de longe e ao mesmo tempo, sem afetar negativamente a taxa de quadros por segundo ou a qualidade visual. Porém, em uma plataforma menos potente, é preciso fazer otimizações, como diminuir o número de objetos sendo mostrados na tela ao mesmo tempo.

Uma das maneiras de fazer isso é realizar *Distance Culling*, ou seja, renderizar apenas os objetos que estejam mais perto do que um certo valor de distância da câmera. Com os objetos mais distantes escondidos, a performance é melhorada significativamente, apesar de perder os detalhes mais distantes, os quais estariam presentes em uma plataforma com maior capacidade de processamento. Podemos ver uma cena sem *Distance Culling* na **Figura 1**, e a mesma cena, na mesma posição de câmera, mas com *Distance Culling* ativado na **Figura 2**.

Figura 1 - A Hat in Time, capturado dentro do jogo, sem *Distance Culling*

Figura 2 - A Hat in Time, capturado dentro do jogo, com *Distance Culling*

Podemos ver na **Figura 1** que vários objetos distantes são renderizados. Alguns destes objetos, como árvores, personagens, decorações e detalhes visuais, não estão presentes na **Figura 2**, tornando mais leve o processamento dos gráficos. A distância a partir da câmera que dita quais objetos serão omitidos, e quais objetos classificados por tamanho devem ser omitidos são escolhas que variam a depender da necessidade de otimização da plataforma.

1.2. Estrutura

Os demais Capítulos deste trabalho possuem a seguinte estrutura:

Capítulo 2. O Problema: Especificação do problema encontrado dado a contextualização da introdução. Também será explicada a maneira que este trabalho aborda o problema.

Capítulo 3. Estado da Arte: Neste capítulo são apresentados estudos e trabalhos de técnicas e conceitos pertinentes a este trabalho, como dicas de pontos a se tratar no desenvolvimento de jogos 3D, no campo de otimizações, sacrifícios e visuais.

Capítulo 4. Proposta de solução: Aqui falaremos como pretendemos tratar o problema apresentado, com o objetivo de mostrar como certos métodos podem fazer com que a experiência em plataformas limitadas em processamento não sejam tão afetadas negativamente.

Capítulo 5. Estudo de Caso: Neste são descritos três casos reais de jogos já lançados em que o conceito deste trabalho pode ser analisado, negativa ou positivamente, e é demonstrado como as sugestões podem ser aplicadas e assim ajudar com o problema descrito.

Capítulo 6. Experimentos: Neste capítulo descreveremos com detalhes as ferramentas utilizadas, os materiais para teste e a abordagem das técnicas propostas na prática.

Capítulo 7. Resultados: Iremos apresentar os resultados de acordo com o que se obteve a partir do experimento, e resultados extras obtidos por observação durante a aplicação dos testes.

Capítulo 8. Conclusão: Por fim, iremos analisar o que pode se aproveitar dos resultados, observar quais foram as limitações, e discutir sobre trabalhos futuros.

2. O Problema

2.1. Especificação

Após feitas as escolhas de balanceamento entre performance e qualidade gráfica, espera-se que o resultado seja um jogo que executa bem na plataforma para qual foi projetado, tendo sido, ou não, sacrificados detalhes visuais para que a experiência seja de alta qualidade. Porém, após feitas essas decisões de projeto, e caso este já esteja em desenvolvimento avançado ou lançado, há casos em que se decida que o jogo será lançado também para outras plataformas. Em várias destas situações, por razões de tempo e gastos, não é prático recomeçar o desenvolvimento, pois como já se tem o ambiente pronto para a plataforma inicial, é mais prático adaptar o produto existente para o novo lançamento. Isso normalmente se refere ao processo de *Porting*, ou Portabilidade do jogo.

O problema que surge deste processo é que graças à decisão prévia de plataforma inicial, as técnicas escolhidas e parâmetros utilizados para balancear performance e qualidade visual não são aplicáveis à nova plataforma em que o produto será lançado. Por isso, é preciso fazer um novo estudo de decisões para melhorar a experiência final nesta nova plataforma escolhida.

Se a nova plataforma tiver maior capacidade de processamento gráfico, as escolhas de otimização para a plataforma inicial são desperdício de capacidade gráfica, pois é possível ter uma qualidade visual melhor sem a necessidade de fazer tantos sacrifícios. Já no caso da nova plataforma ter menor capacidade, surgem novos problemas, pois como as otimizações já foram feitas no desenvolvimento da plataforma original, não restam opções para melhorar a performance a não ser diminuir a qualidade gráfica, os detalhes visuais.

Entre estes, o mais comum é o segundo, quando normalmente os jogos ganham uma nova versão para uma plataforma móvel (como celulares, tablets) ou consoles portáteis. Por não terem o mesmo poder de processamento das plataformas originais, os jogos nestas plataformas, se fossem transferidos sem nenhuma mudança de balanceamento gráfico, iriam obter níveis de performance inaceitáveis para jogadores.

2.2. Abordagem

Neste trabalho nós iremos focar no caso em que a nova plataforma decidida para o produto tenha menos capacidade de processamento que a original. Como dito na seção **2.1**, já que as otimizações já foram feitas anteriormente, é preciso sacrificar qualidade gráfica e detalhes visuais. Por isso, se tenta sacrificar partes em que o impacto visual seja o menor possível.

Por mais que se tente minimizar este impacto, normalmente é inevitável que a diferença seja perceptível. Assim, certas escolhas precisam ser feitas, tanto com o objetivo de repensar decisões anteriores, como para adotar novas opções de design para ajudar na diminuição deste impacto. Essas escolhas nem sempre são triviais, e dependem de cada projeto com seu estilo gráfico e público alvo único. Apesar disso, neste trabalho, propomos algumas maneiras gerais de obter essas melhorias, e que podem ser ajustadas para cada caso.

3. Estado da Arte

Com o avanço tecnológico dos últimos anos, a produção de jogos digitais 3D tem se tornado cada vez mais complicada. Técnicas como *Culling***[1]**, *LODs***[2]**, *Instancing*(Instanciamento de vários objetos idênticos para serem renderizados em uma única chamada na GPU), *Cubemaps***[3]** e outros, já são utilizados em grande escala desde os primórdios dos jogos em três dimensões.

Para integrar nosso objetivo, além de técnicas extras de otimização, iremos precisar também de métodos para mitigar o impacto negativo causado no visual, como por exemplo *Dithering***[4]** para compensar a diminuição da distância de renderização, ou *Depth of Field***[5]** para compensar a simplificação de modelos ou uso de *Imposters***[6]** em objetos distantes.

O trabalho de Chehimi**[7]** mostra alguns pontos específicos para se prestar atenção na hora de desenvolver jogos para plataformas *mobile*, que geralmente possuem menos poder de processamento. Ele lista várias preocupações quanto a otimização de materiais, de modelos, de código, de representações visuais e de modelos de processamento de geometria. É uma listagem muito útil para este trabalho, já que pode servir de ponto de partida para observar partes do processamento gráfico que serão atingidas para se conseguir estabilidade de performance, e onde poderemos observar se há a possibilidade de alguma técnica de compensação.

Dentre os paradigmas de otimização descritos por Chehimi**[7],** Seção 4.4, os 6 primeiros são focados em código, e não são cobertos por este trabalho, já que em teoria são otimizações já feitas na plataforma inicial, e não pode se melhorar mais. Os paradigmas 7 e 8 se preocupam com a quantidade de polígonos e primitivas detalhadas, o que é algo que além de poder ser melhorado alterando os *assets*, também pode ser complementarmente resolvido com técnicas a serem descritas neste trabalho, como alterar o *Draw Distance*, fazendo assim com que os objetos que estejam mais longe da câmera não precisem ser renderizados. Ao fim, nos paradigmas 9 e 10 se fala em escalar as texturas e comprimir os arquivos, graças às telas de dispositivos móveis serem em geral menores, e não serem impactados fortemente pelo aumento da compressão. Similarmente, iremos descrever neste trabalho sobre a diminuição de *Screen Percentage*, o que não é um problema grande em telas menores, como em portáteis.

Hosseini**[9]** propõe um framework para a transferência de texturas 3D que sejam adquiridas durante o jogo, como para jogos on-line em que texturas podem mudar em tempo real de acordo com um servidor ou outro jogador. O sistema proposto visa poupar o máximo de processamento e energia possível. Tal sistema pode também ser muito útil nesses casos para melhorar a performance durante jogos on-line.

O sistema RAVEN**[10]** foi desenvolvido para se aproveitar de similaridades entre quadros (*frames*) próximos uns dos outros, para fazer pulos estratégicos e poupar o processamento da GPU de dispositivos móveis, que é o que utiliza mais força do sistema destes dispositivos. Os autores buscam uma diminuição mínima da qualidade visual para poder obter uma melhora significativa no gasto de energia e processamento.

O trabalho de Koskela, T.**[12]** demonstra um grande conjunto de técnicas para ambientes 3D que variam o foco entre simplificações, otimizações, e formas de demonstrar os objetos. Também mostra um esquema que demonstra as três preocupações quando desenvolvendo este tipo de aplicações, o qual pode ser visto na **Figura 3**. Os três pontos são Performance, Qualidade Visual e Consumo de Energia, e cada projeto vai ter uma relação de equilíbrio diferente entre os três, que depende do estilo e objetivo de design de cada um.

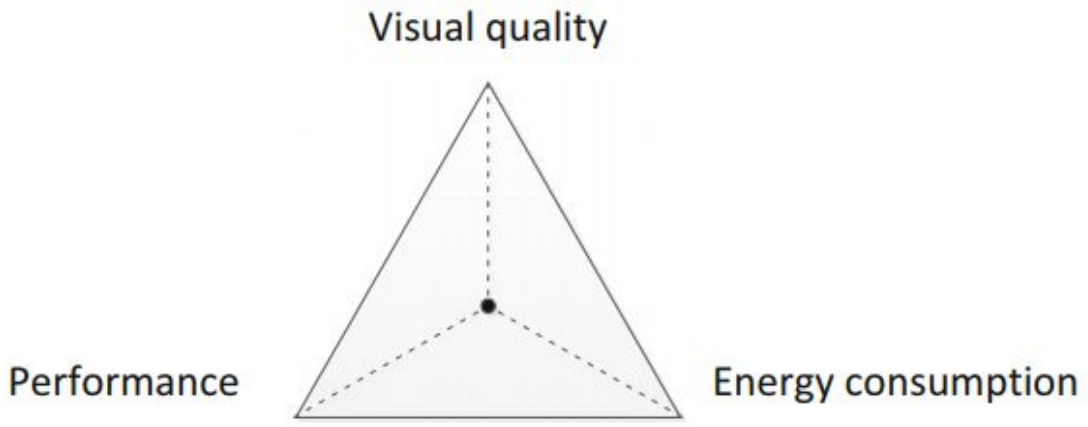

Figura 3 - Esquema de equilíbrio para dispositivos móveis, de Koskela, T.[12]

Apesar dos três pilares serem importantes no desenvolvimento de ambientes 3D em dispositivos móveis, neste trabalho nós iremos focar no balanceamento entre dois deles: Performance e Qualidade Visual.

4. Propostas de Solução

Este trabalho tem como objetivo auxiliar no desenvolvimento de jogos 3D, com foco no movimento de portabilidade para plataformas menos poderosas e a adaptação a estas, ao mesmo tempo amortecendo o dano visual causado pela diminuição na qualidade gráfica. Queremos maximizar a performance, ainda mantendo um padrão de conforto na jogabilidade.

Para atingir este objetivo, iremos:

- Propor métodos que melhorem substancialmente a performance, sem grandes preocupações com o efeito na qualidade visual
- Após isso, mostrar técnicas para mitigar o impacto dos métodos propostos sobre a qualidade visual do jogo.

As sugestões aqui feitas não devem ser seguidas às cegas, sem que antes ocorra um estudo de caso, pois elas podem alterar o estilo visual do jogo, o que pode não ser desejado. Escolher quais as melhores técnicas a usar é decisão de projeto.

Os principais métodos de otimização e suas técnicas de amortecimento sugeridos são serão descritos nas próximas seções deste capítulo. Estas propostas de solução se baseiam no ambiente de desenvolvimento UE4 - Unreal Engine 4. Os conceitos discutidos não são exclusivos desta ferramenta e podem ser aplicados em outros ambientes. Os detalhes sobre como a UE4 foi utilizada para este trabalho se encontram no **Capítulo 6. Experimentos** deste mesmo.

4.1. Draw Distance

Diminuir a distância máxima de renderização *(Draw Distance)* por objeto, e a distância das trocas de nível de detalhe (*LOD***[2]**), para renderizar menos objetos no total, e apenas aumentar os seus detalhes quando mais perto deles.

Os efeitos negativos serão diminuídos com a técnica de *Dithering***[4]**, que realiza uma troca suave e filtrada entre os níveis, tornando menos perceptíveis as trocas de modelo que serão feitas mais perto do ponto de visão, pela diminuição do *Draw Distance*. Inicialmente esta foi implementada em lógica de material, como demonstrado na **Figura 4**. Esta seção é o foco deste trabalho, por considerarmos produzir os resultados mais significativos. Portanto as técnicas serão descritas com mais detalhes que as demais.

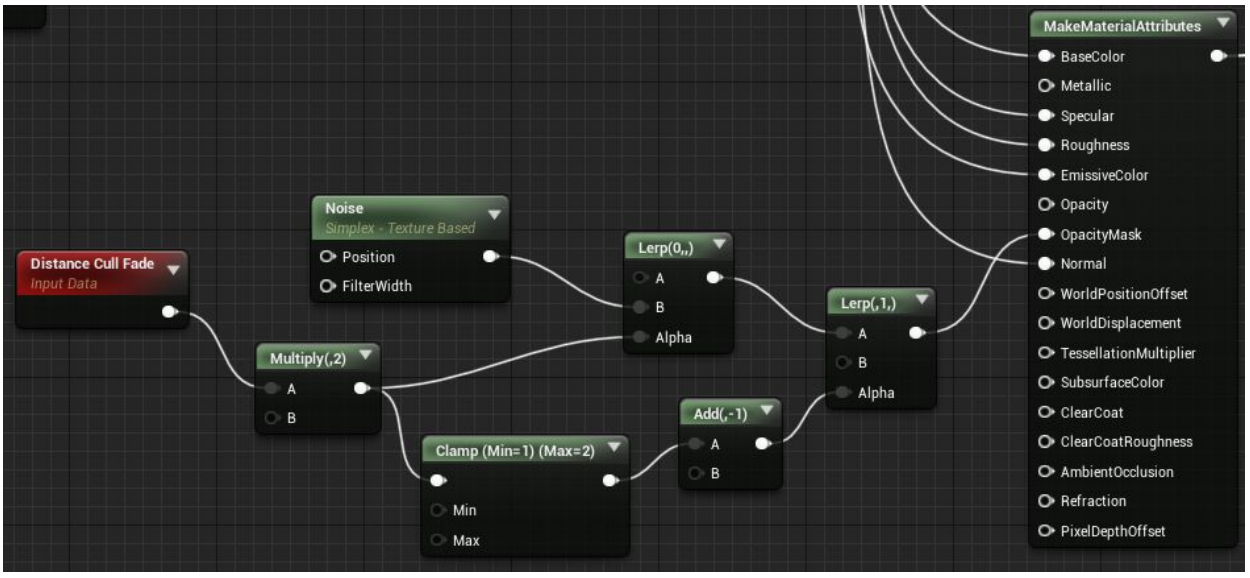

Figura 4 - Lógica em material na UE4 para reproduzir o efeito *Dithering***[4]** individual

O parâmetro *Distance Cull Fade* equivale a um valor numérico entre 0 e 1 que representa a transição entre renderizado e não renderizado, para um objeto atravessando seu limite de distância de desenho (*Draw Distance*). Este valor é aqui utilizado com uma textura de ruído aleatório (*Noise*) para produzir um efeito de dissolução no objeto que fizer a transição entre renderizado e escondido. Algumas etapas deste efeito podem ser vistas na **Figura 5**. Quando um objeto passa pelo *Dithering***[4]**, leva 0.3 segundos para completar o processo, porém este valor pode ser alterado no código da Engine.

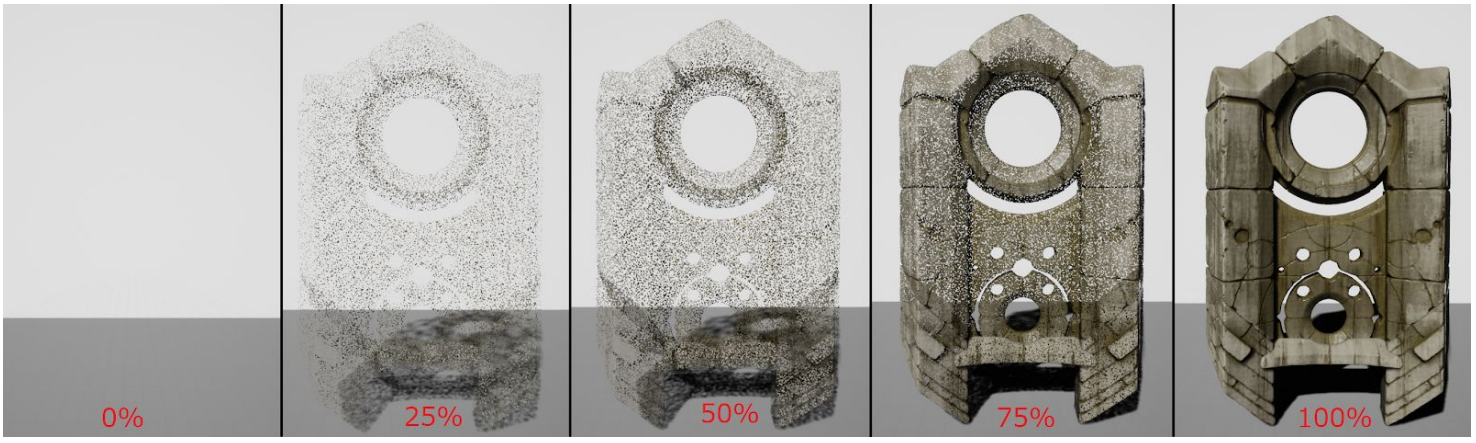

Figura 5 - Objeto com efeito *Dithering***[4]** em valores diferentes

Este efeito de dissolução é o que suaviza a transição entre visível e escondido, e o mesmo para o caminho contrário. Outra opção para atingir um resultado similar seria simplesmente fazer uma transição de transparência. Porém, foi escolhido o uso do efeito *Dithering***[4]**, pois este, apesar de muito parecer, não utiliza de nenhum cálculo de transparência entre pixels, ele apenas mascara partes do material, de acordo com a textura de ruído, para completamente visível, ou totalmente invisível, dando assim uma aparência muito similar à de transparência.

O motivo desta escolha é a natureza dos materiais, pois aqueles que sejam do tipo transparente (*Translucent*) acarretam em um peso maior no processamento e dificuldade em usar sombreamento e iluminação, enquanto os mascarados (*Masked*), são mais leves, trazendo assim o objeto a uma necessidade de processamento próxima a de objetos opacos. Na **Figura 6** podemos ver uma comparação do efeito em transparência, e o efeito *Dithering***[4]**.

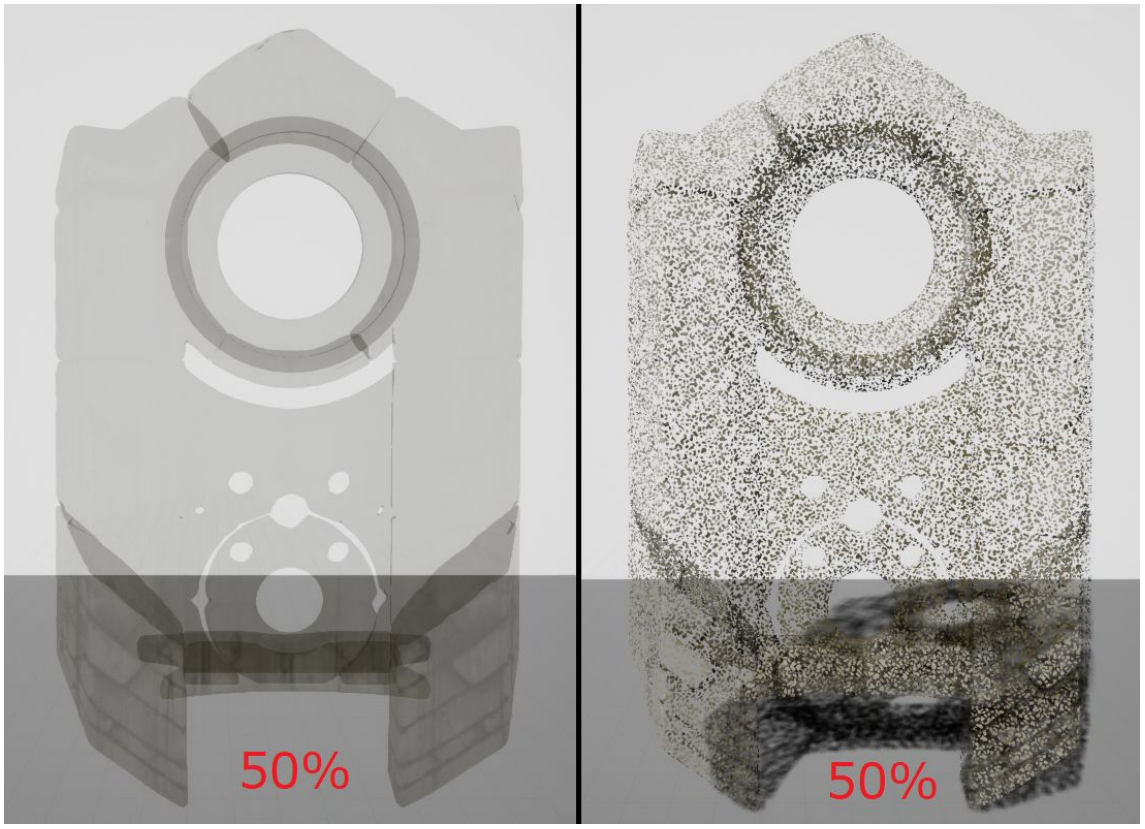

Figura 6 - Comparação entre as técnicas de transparência(esquerda) e de *Dithering***[4]**(Direita).

Outro ponto a se destacar sobre o uso da técnica *Dithering***[4]**, é que se formos a utilizar em conjunto com a técnica de *Depth of Field***[5]**, a diferença entre o *Dithering***[4]**

e a Transparência se torna ainda menos visível, principalmente por estar mais ao fundo da tela. Os pixels visíveis do objeto que seriam vistos com artefatos da textura de ruído ficam mais suaves e mais próximos do efeito de transparência após serem desfocados.

Figura 7 - Comparação entre o uso da técnica *Dithering***[4]** sem DoF(Esquerda) e com DoF(Direita), vistos de longe.

Este será o método utilizado no experimento, o qual é descrito no capítulo 6 deste trabalho. Utilizando lógica de materiais desta maneira, o efeito é alcançado, porém precisa ser aplicado individualmente a cada material, e adiciona complexidade, afetando a performance mais do que o desejado para uma técnica de mitigação de impacto visual. Ainda que seja uma vantagem utilizar deste método, podemos melhorar mais ainda, para sacrificar menos performance.

Por isso, há outra maneira de fazer este efeito, e que tornará o trabalho necessário para usá-lo muito menor, além de ser mais eficiente. Nós podemos modificar o código nativo dos *shaders* para incluir essa funcionalidade. Na UE4, estes *shaders* são descritos em linguagem HLSL, junto ao código nativo da Engine. O código da **Figura 8** faz um teste de acordo com o estado do *Dithering***[4]** e da textura de ruído. A função é chamada para um pixel específico, por ser um shading em espaço de tela. Para cada pixel, fazemos este teste, e a função “clip” faz com que descartemos o desenho do pixel que tenha valor abaixo de 0 no argumento.

```
void DoDithering( float2 ScreenPos, float DitheringOpacity)
{
   half3 Noise = tex2D( NoiseTexture, ScreenPos.xy).xyz;

   clip( Noise.x - ( 1 - DitheringOpacity ) );
}
```

Figura 8 - Código em HLSL para aplicar a máscara de textura usando *Dithering*[4]

Ao ser chamado, este código aplica para os pixels de uma primitiva específica. Naturalmente esta primitiva será a do nível de detalhe (LOD) selecionado de acordo com a distância da câmera para o objeto. Como aqui estamos tratando de transições no limite de distância de renderização do objeto, tipicamente isto acontece apenas no último LOD. Nós também queremos utilizar o conceito de *Dithering*[4] para as transições entre LODs, por também custar bastante ao processamento. No caso do UE4, esta é uma função nativa, chamada "*Dithered LOD Transition*", como opção nas propriedades de cada material. Porém, iremos demonstrar neste trabalho como fazer também HLSL. Na **Figura 9** fazemos o preparatório dos argumentos em C++ para usar no HLSL. Como queremos fazer uma transição entre níveis, precisamos que ambos sejam renderizados por um breve momento, para podermos fazer a troca. Para uma das duas primitivas, nós invertemos a textura de ruído, para justamente termos uma completando a outra, ao invés de transparência nos pontos vazados.

```
DitheringSettings.x = DitheringOpacity;

if( IsInverted )
{
   DitheringSettings.y = -1.0f; // Noise scale
   DitheringSettings.z = 1.0f; // Noise bias
}
else
{
   DitheringSettings.y = 1.0f; // Noise scale
   DitheringSettings.z = 0.0f; // Noise bias
}
```

Figura 9 - Preparando valores em C++ para serem usados no HLSL

A decisão de quais primitivas usam o padrão invertido se dá com a paridade da ordem LOD. Dessa forma sempre será feita a transição cruzada. Com os argumentos preparados, utilizamos uma versão alterada do código em HLSL para levá-los em conta na hora do cálculo e decisão do descarte ou não dos pixels, como demonstrado na **figura 10**.

```
void DoDithering( float2 ScreenPos, float3 DitheringSettings)
{
   const float DitheringOpacity = DitheringSettings.x;

   const float NoiseScale = DitheringSettings.y;
   const float NoiseBias = DitheringSettings.z;

   half3 Noise = tex2D( NoiseTexture,  ScreenPos.xy ).xyz;
   float ScreenFactor = NoiseBias + NoiseScale * Noise.x;

   clip( ScreenFactor - ( 1 - DitheringOpacity ) );
}
```

Figura 10 - Utilizando os argumentos de modificação de ruído em HLSL

Com isto, podemos reduzir drasticamente as distâncias em que são feitas trocas de LOD, pois serão menos perceptíveis e mais elegantes, sendo assim mais um fator que melhora a performance com impactos menos perceptíveis no visual.

Outras aplicações a se considerar são implementar o *Dithering***[4]** também considerando transições dos modelos com *Imposters***[6]** e *Decals***[11]**. Além disso, pode-se realizar outras melhorias no *Dithering***[4]**, como utilizar a velocidade da câmera para determinar quantos frames o efeito levará para terminar, assim tornando ainda menos perceptíveis as transições. Quanto mais rápida, menos tempo a transições devem demorar.

4.2. Screen Percentage

Diminuir o Percentual de Tela *(Screen Percentage)*, que dita em qual resolução o jogo deve renderizar para depois ser escalado para a resolução da tela do dispositivo. Isto melhora drasticamente a performance de um jogo, por tornar necessário menos pixels serem processados. Porém o impacto visual é grande.

Para reduzir tal impacto significativo, pode-se utilizar Resolução Adaptativa. Com esta, podemos realizar trocas de resolução durante a execução do jogo. Baseia-se em uma heurística para tentar prever se o processamento no futuro próximo será mais pesado ou mais leve. Com base nisso, decide-se qual a resolução de renderização, dinamicamente. A **figura 11** de um artigo da Intel, por doug-binks[13] esquematiza o funcionamento dessas trocas de resolução.

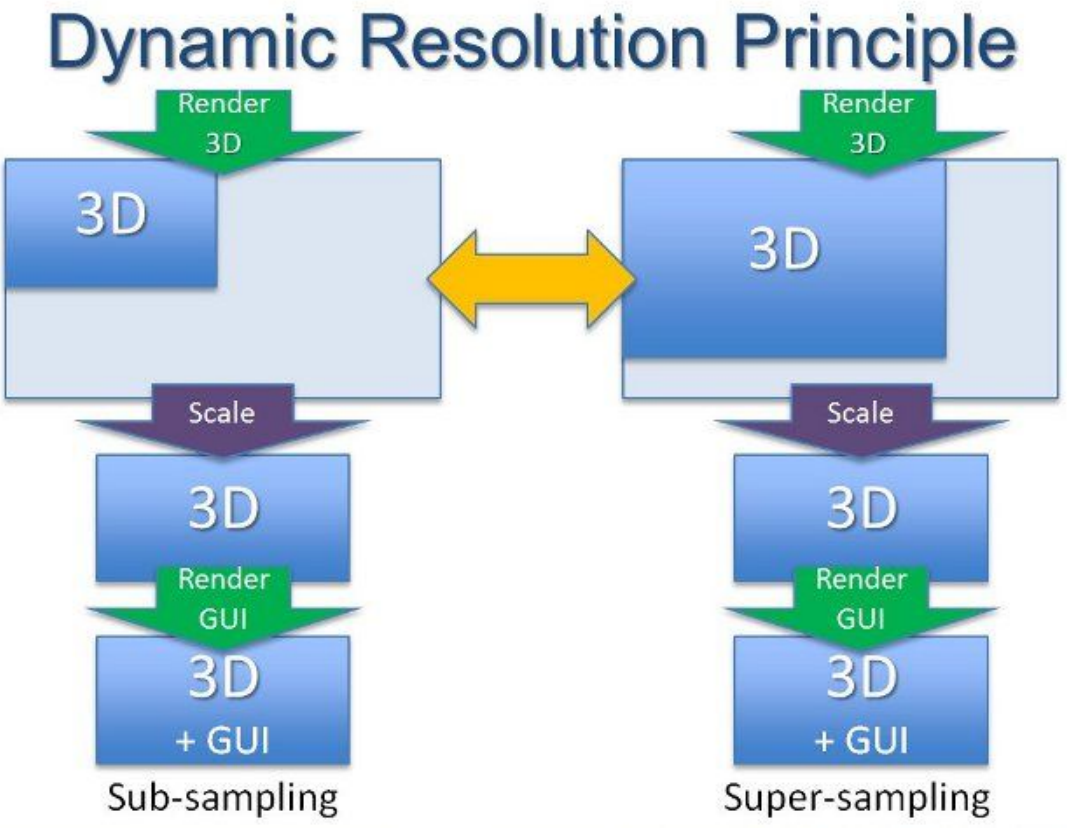

Figura 11 - Esquema de Resolução Adaptativa por doug-binks**[13]**

Como estamos tratando de plataformas menos potentes, normalmente não é indicado utilizar Super-Sampling, a não ser que se tenha certeza de que é possível fazer isso sem ferir a performance. Para isto, as heurísticas precisam ser bem definidas e considerar vários detalhes como também o estado de outros sistemas de otimização, como o número de objetos em estado de transição na técnica de *Dithering***[4]** descrita na seção anterior. Normalmente interfaces de usuário não são afetadas pelo uso de Resolução Adaptativa, pois são renderizadas após todo o processo de renderização 3D, a não ser que sejam interfaces inseridas no mundo. Também importante citar que o sub-sampling pode ocorrer independentemente na altura e largura da tela. Ou seja, para uma tela de 1280 por 720 por exemplo, podem ocorrer momentos em que a resolução de renderização seja 1024 por 720 (Mantendo altura final), ou 1024 por 648 (Mantendo a largura em relação à anterior), como é no caso de Doom**[18]** no Nintendo Switch.

Outra técnica utilizada neste caso é a de *Depth of Field***[5]**, assim suavizando a visão do que estiver mais longe, que seria o mais afetado pela diminuição de percentual de tela. No melhor dos mundos, combina-se DoF com Resolução Adaptativa, porém essa é uma tarefa complexa, então nem sempre será possível. Caso não possa se combinar, é preferível manter Resolução Adaptativa mesmo sem DoF, por ser uma técnica dinâmica, de acordo com a heurística definida.

4.3. Imposters

Utilizar *Imposters***[6]** para substituir o modelo de nível de detalhe mais baixo dos objetos por um único Sprite 2D, gerado a partir do modelo 3D original, trocando o processamento de todos os polígonos deles por uma textura bidimensional em apenas um ou poucos polígonos. Se cria uma lista de Sprites 2D, um para cada ângulo de visão do objeto, variando-se a quantidade, e portanto a qualidade em movimento, por escolha.

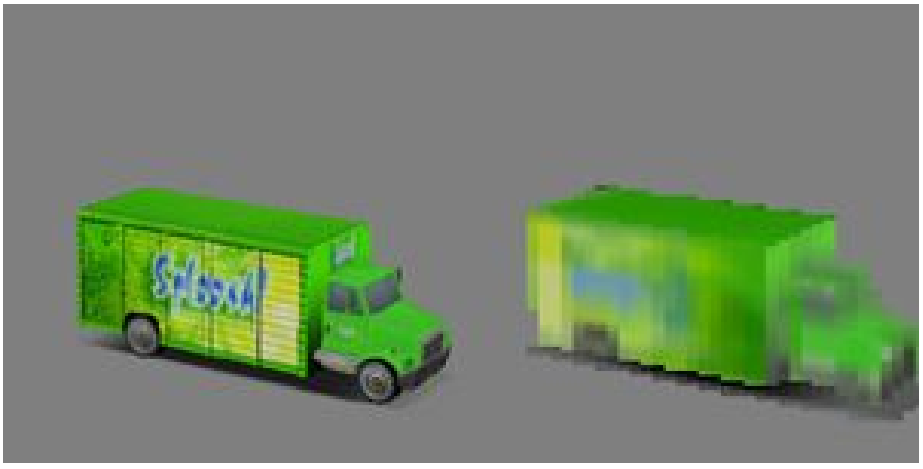

Figura 12 - Modelo real (Esquerda) e o *Imposter***[6]** criado(Direita)

Também podemos reduzir os efeitos negativos deste método com a técnica de *Depth of Field***[5]**, que de certo grau tira de foco os objetos que estiverem mais longe. Tipicamente isso inclui todos os objetos que estiverem na representação de *Imposter***[6]**, pois estes normalmente são utilizados para substituir o nível de detalhe (LOD) mais distante. Outra opção é adicionar um novo nível de detalhe para utilizar o *Imposter***[6]**, assim teremos como estender o *Draw Distance* sem perder a qualidade de um LOD do modelo. Além do DoF também pode se usar névoa para disfarçar os *Imposters***[6]**.

4.4. Precomputed Visibility

Diferentemente de *Distance Culling*, a técnica *Precomputed Visibility*[15] dita quais objetos devem ser renderizados ou não a partir do ambiente específico que a câmera esteja. Na **figura 13** podemos ver diversas células azuis geradas para particionar o mundo em ambientes em que se espera ser navegável pela câmera. Cada uma destas células está relacionada com outras áreas, que englobam os objetos no mundo. Por exemplo, se a câmera estiver no ponto "A" demonstrado na figura, é esperado que os objetos englobados pela região em vermelho, que estão do outro lado da parede

bloqueando visão, não sejam renderizados, já que eles não deveriam ser vistos. Este tipo de Culling não aconteceria apenas com Distance Culling, pois estes objetos estão próximos da câmera, e portanto podem até renderizar no LOD mais detalhado, causando assim um impacto desnecessário na performance.

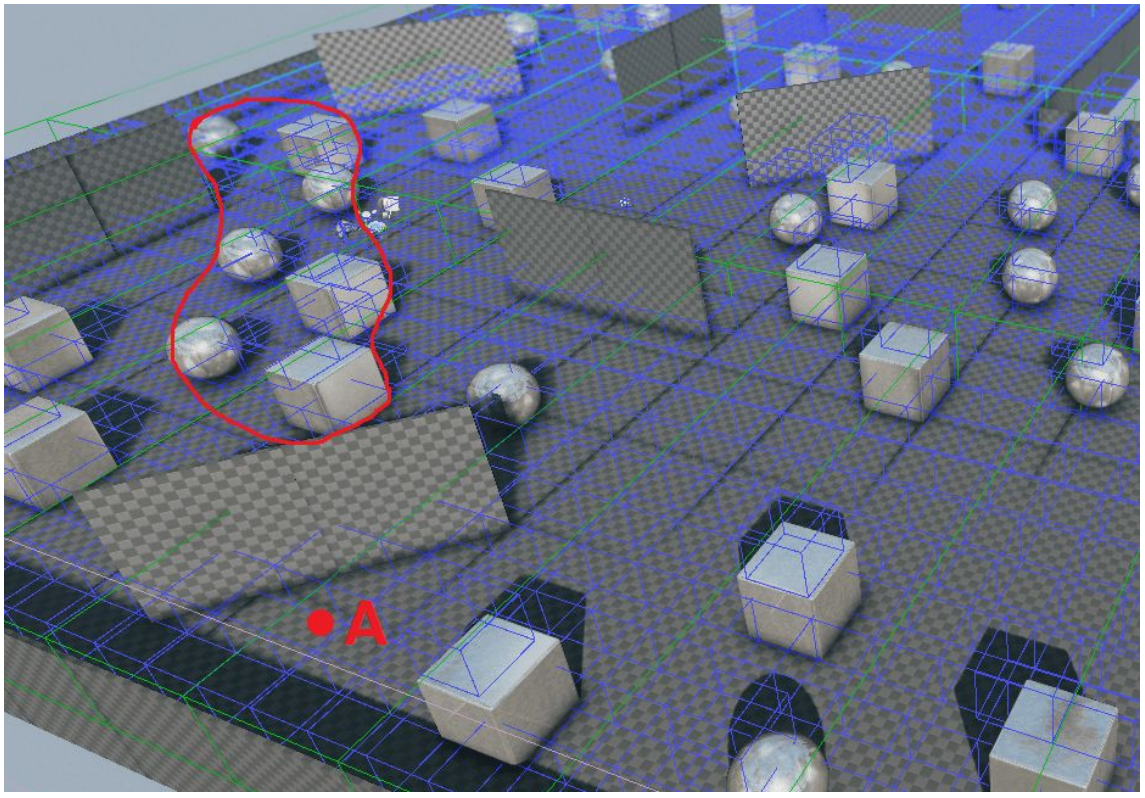

Figura 13 - Esquema retirado do guia no fórum da plataforma Steam[17]

Ao contrário do *Distance Culling*, não podemos confiar sempre no DoF ou no *Dithering***[4]**, pois muitas vezes os objetos a aparecerem estão muito próximos, apesar de bloqueados por algo no caminho. Por isto, como DoF ocorre no campo de fundo da visão, neste caso uma técnica recomendada para se utilizar é a de *Motion Blur*[23], já que normalmente a transição para visão destes objetos se dá com o movimento de câmera, com algum objeto que esteja na frente se movendo, ou com algo que resulta em translação do objeto na posição da tela. Desta forma, caso algum objeto mude de invisível para renderizado, será menos perceptível no campo de visão distorcido.

4.5. Reflections

Reflexos calculados em tempo real são muito desejados em jogos de estilo visual realista, porém são bastante complexos e acarretam em alto custo de performance. Na **Figura 14** podemos ver que os reflexos que existem na versão para PlayStation 4 do jogo Doom**[18]** foram retirados para a versão do Nintendo Switch. Os reflexos que restaram são bem menos aparentes.

Figura 14 - Retirada de um vídeo de análise comparativa do jogo Doom**[18]** entre diversas plataformas

Para não perder os reflexos de ambiente, podemos aplicar a técnica de *Cubemaps***[3]**, que captura o ambiente ao redor de um ponto e armazena a informação de imagem em textura na forma de um cubo. Estas informações podem ser usadas para simular reflexos. Por serem informações criadas antes da execução do jogo, apenas fazem reflexo do ambiente estático em volta, portanto não mostrará o reflexo de personagens e efeitos dinâmicos. Na **Figura 15** podemos ver uma cena com reflexos feitos inteiramente por *Cubemaps***[3]**.

Figura 15 - Reflexo de em uma sala feito inteiramente com *Cubemaps***[3]**

4.6. Interior Reduction

Muitas vezes, ambientes em jogos possuem muitos detalhes, mas não são espaços acessíveis pelos personagens ou pela câmera do jogo. Estes espaços podem possuir muitos polígonos que nunca entrarão em contato com outros objetos do jogo. Portanto, com o objetivo de reduzir o número de polígonos desnecessários, pode-se renderizar apenas um *Sprite* 2D com um desenho representando o interior da sala, similar ao conceito de *Imposters***[6]**, mas para representar vários objetos ao invés de apenas um.

Apesar de ajudar na performance, apenas um desenho para representar o ambiente muitas vezes é muito simples, especialmente se tratando de ambientes que em algum momento são visíveis de perto. Assim, desejamos simular o ambiente interno, dando a sensação de profundidade, sem utilizar a geometria original durante a execução. Para isso, similarmente à redução de impactos visuais da remoção de Reflexos, como descrito na seção anterior, podemos utilizar *Cubemaps***[3]** para simular ambientes internos. Poucos polígonos são utilizados para dar a impressão de profundidade, como demonstrado na **Figura 16**, onde se pode ver que a profundidade não é real, pois o espaço atrás do plano onde a sala é vista está vazio. Não é possível realizar efeitos dinâmicos dentro da sala, porém é uma representação mais próxima da realidade do que um desenho 2D.

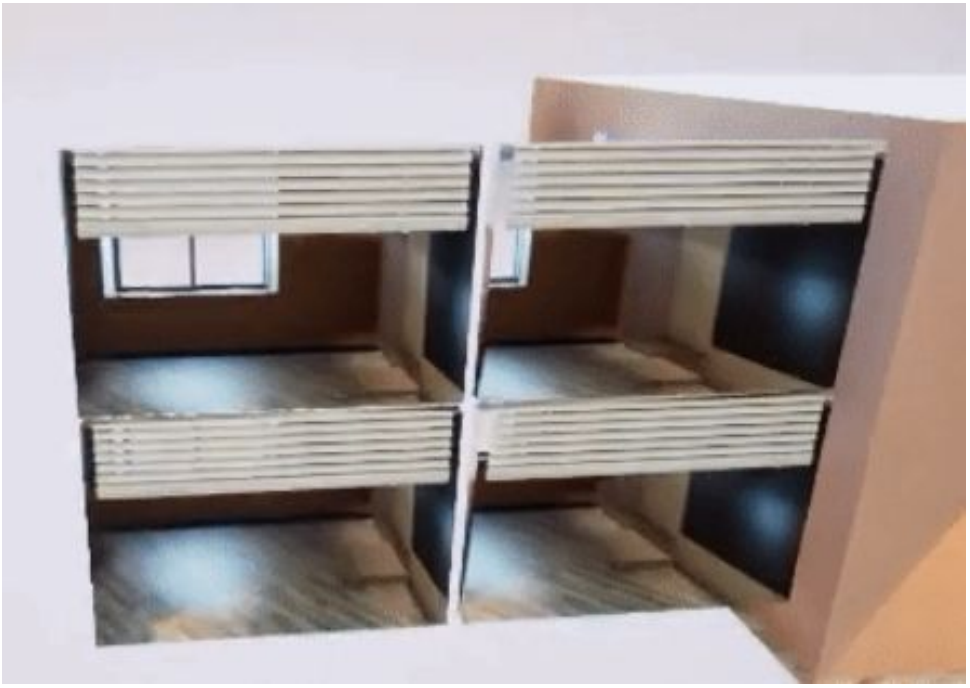

Figura 16 - Interiores de sala utilizando *Cubemaps*[3]

4.7. Outros

Além dos métodos descritos nas seções deste capítulo, muitos outros podem ser considerados, como:

- Diminuição da frequência de atualização para objetos dinâmicos distantes
    - Pode ser feito interpolação entre as posições para simular fluidez
- Filtragem da qualidade das sombras e texturas de acordo com a distância
    - Esta transição pode ser feita de forma gradual
- Menor abundância de folhagem
    - Mudanças de textura de solo para disfarçar tal diminuição.
- Anti-Aliasing para ajudar na diminuição de resolução (*Screen Percentage*), junto a ter uma Resolução Adaptativa

5. Estudo de Caso

Neste capítulo, analisamos a relação do problema e das propostas de solução com jogos já lançados e inseridos no mercado, assim já tendo opiniões públicas, bem como análises e críticas profissionais. Veremos que em parte dos casos, os jogos utilizam das técnicas descritas, e conseguem obter sucesso no balanceamento de performance e fidelidade visual, enquanto outros sofrem neste equilíbrio por não utilizarem, ou por utilizarem pouco ou de maneira errada.

5.1. RiME

Com o objetivo de analisar o problema explorado neste trabalho em instâncias reais, iremos observar o caso da versão para a plataforma Nintendo Switch feita do jogo *RiME***[8]**. Inicialmente o jogo havia sido lançado para as plataformas Microsoft Windows, XBox One e Playstation 4. *RiME* foi desenvolvido usando a Unreal Engine 4.

A versão para Nintendo Switch deste jogo foi lançada seis meses após o lançamento das demais plataformas. Como o Nintendo Switch tem menos poder de processamento, algumas adaptações precisaram ser feitas, como comentado neste trabalho sobre tais situações.

Nas versões originais do jogo, pouco se vê em questão de transições bruscas de nível de detalhe (LODs), e não é possível perceber objetos aparecendo ou desaparecendo da tela. Isto é ótimo, dado que nestas plataformas originais, o jogo na maior parte do tempo executa em níveis aceitáveis de performance. Apesar disso, já se nota na versão para Windows que não há instâncias de uso de *Imposters***[6]**, o que seria uma ótima opção nesse caso, por se tratar de um jogo de ritmo lento, então não haveriam transições bruscas com movimentos rápidos de câmera, dando liberdade para mostrar objetos que estejam longe com bem menos detalhes. Em geral para o Windows, isto não é um problema grande.

Porém, no Nintendo Switch a performance aparece quase sempre em níveis baixos, com taxas de quadros (*frame rate*) baixas, e várias instâncias de congelamento breve. As análises profissionais no aclamado *Metacritic***[20]** denunciam grandes problemas de performance, e a diferença de média de notas para as outras plataformas é de pelo menos 14 pontos percentuais. Podemos ver lugares no jogo como no da **Figura 17**, que apresentam uma grande quantidade de objetos renderizados ao mesmo tempo.

Figura 17 - Visão retirada do jogo RiME**[8]**

Após alguns meses do lançamento da versão para Nintendo Switch, os desenvolvedores do jogo enviaram uma atualização que melhorou a taxa de quadros e estabilidade, alterando *Draw Distance* e *Screen Percentage*. Porém a performance e qualidade visual ainda poderiam melhorar aproveitando mais os métodos.

	Utiliza Método?	**Reduz impactos?**
Draw Distance	Muito pouco, tanto no *Culling* quanto na transição entre níveis de detalhe	Apesar de utilizar pouco do método, realiza *Dithering* no *Culling*. Trocas de níveis de detalhe são abruptas
Screen Percentage	Sim, a resolução de renderização é bem menor que a resolução final	Não possui resolução adaptativa. Também não usa Depth of Field
Imposters	Não, apesar do grande potencial no estilo de jogo	-Não se aplica-
Precomputed Visibility	Sim, em algumas áreas que portas bloqueiam	Não utiliza motion blur nem outro efeito. Transição brusca e aparente
Reflections	Apenas *Cubemaps*	Já usa *Cubemaps*
Interior Reduction	-Não se aplica-	-Não se aplica-

Tabela 1 - Relação das técnicas usadas no jogo RiME**[9]**

Alguns outros pontos negativos em RiME**[8]** incluem:

- Sem diminuição da frequência de atualização para objetos dinâmicos distantes.
- Sem filtragem da qualidade das sombras e texturas de acordo com a distância.
- Folhagem completamente estática.

5.2. Doom

Contrapondo o caso do jogo RiME**[8]** descrito na seção anterior, a versão de Doom**[18]** para Nintendo Switch foi muito bem recebido, tendo notas melhores no *Metacritic***[20]** e comentários positivos de críticos profissionais, com pontos negativos por fidelidade visual pior que o original, porém sem grandes problemas de performance.

Figura 18 - Jogo Doom**[18]**, retirado de Doom Switch Review - GameSpot

	Utiliza Método?	**Reduz impactos?**
Draw Distance	Pouco *Culling*, muita transição entre níveis de detalhe	Apesar de utilizar pouco do método, realiza *Dithering* no *Culling*. Trocas de níveis de detalhe também suaves
Screen Percentage	Sim, a resolução de renderização diminui em relação à da final	Sim, possui resolução adaptativa. E usa bastante Depth of Field, dinâmico
Imposters	Possui poucos ambientes à distância, mas quando tem, usa, ou usa versões simples de modelos.	Utiliza névoa quando são muito longe
Precomputed Visibility	Sim, o jogo é em maior parte em salas, então possui vários volumes	Motion Blur é fortemente utilizado, e volumes bem localizados.

Reflections	Sim, Reflexos em tempo real são removidos ou diminuídos	Sim, *Cubemaps* em alguns lugares, com auxílio de luzes adicionais.
Interior Reduction	Sim, para muitos compartimentos, ao invés de salas	Sim, *Cubemaps* para compartimentos que não se pode alcançar.

Tabela 2 - Relação das técnicas usadas no jogo Doom**[18]**

Outras técnicas implementadas em Doom**[18]** incluem:

- Filtragem da qualidade das sombras e texturas de acordo com a distância, com transição gradual.

5.3. Fortnite

Sendo um dos maiores jogos do mundo no momento**[21]**, Fortnite**[19]** tem por trás uma equipe especializada (da mesma empresa criadora do UE4, Epic Games Inc.) e que se preocupa com a performance do jogo em todas as plataformas lançadas.

Figura 19 - Comparando gráficos de versões de Fortnite**[19]** em plataformas diferentes

Por utilizar fortemente alguns dos métodos propostos, principalmente o referente a *Draw Distance*, tanto em *Culling*, quanto em LODs, e também por aplicar as técnicas

de redução de impacto, Fortnite**[19]** alcança performance estável, e visuais que não deixam a desejar. A Resolução Adaptativa com eixos independentes também funciona para garantir taxa de quadros estável, mesmo que ainda aconteçam algumas quedas momentâneas.

	Utiliza Método?	**Reduz impactos?**
Draw Distance	Bastante *Culling*. Muitas transições de LOD	*Dithering* no *Culling* e nas transições de LOD. Não usa DoF
Screen Percentage	Sim	Sim, possui resolução adaptativa. Não usa DoF
Imposters	Sim, nos objetos mais distantes	Não usa DoF
Precomputed Visibility	-Não se aplica- Construções destrutíveis	-Não se aplica- Construções destrutíveis
Reflections	Sim, Reflexos em tempo real são removidos ou diminuídos	Sim, substituição por *Cubemaps*
Interior Reduction	-Não se aplica- Construções destrutíveis	-Não se aplica- Construções destrutíveis

Tabela 3 - Relação das técnicas usadas no jogo Fortnite**[19]**

Outras técnicas implementadas em Fortnite**[19]** incluem:

- Diminuição da frequência de atualização para objetos dinâmicos distantes, mas sem interpolação (Nas demais versões parece ocorrer interpolação)
- Filtragem da qualidade das sombras e texturas de acordo com a distância, com transição gradual.
- Alternativa ao *Dithering* para o *Culling* de alguns objetos, onde eles começam em escala 0 e crescem rapidamente para o tamanho original do objeto.
- Menor abundância de folhagem, acompanhado por mudanças de textura de solo para disfarçar.

6. Experimentos

Para atingir os objetivos definidos, foi usada a Unreal Engine 4, por ser uma das ferramentas mais robustas para desenvolvimento de jogos 3D, capaz de servir de ambiente para os testes e implementações desejados. Foi criado um projeto com base de código C++. Pode-se também escolher utilizar *Blueprints* (que são scripts visuais) para descrever as classes do projeto, porém foi decidido que não era pertinente, por ser um experimento também de cunho específico quanto à performance.

Além disso, foi utilizado o código fonte original disponibilizado pela Epic Games, Inc. (que possui totais direitos da UE4) no GitHub para compilar a Engine localmente. Isto foi feito para termos acesso a todo o código interno da ferramenta, e tornar possível o estudo e modificação da Engine de qualquer forma que fosse preciso. A partir desta geração local da versão da Engine, o projeto criado fica atrelado à ela, tornando necessário que o projeto em si seja compilado novamente cada vez que a Engine seja modificada em seu código nativo.

Tais modificações de código nativo foram feitas para termos mais liberdade de modificação, que não se teria normalmente com a versão comum. A técnica utilizada modificando o código nativo da Engine foi o efeito *Dithering***[4]** para combater os impactos visuais da diminuição do *Draw Distance*.

Os *assets* utilizados nos experimentos são de pacotes que contêm materiais de alta qualidade a nível de jogos AAA(denominação dada para jogos da mais alta qualidade no mercado), disponibilizados de graça pela empresa Epic Games, Inc., além de outros outros criados durante o experimento.

Para visualização dinâmica da cena criada, foi implementada uma câmera que usa um componente de movimentação em formato de Seno ao longo de um dos eixos. Desta forma, pode se visualizar e avaliar melhor o efeito efeito *Dithering***[4]**, por ser um movimento ortogonal ao limiar de transição dos estados ditado pelo limite de *Draw Distance*. Na Figura 20 podemos ver a cena criada, e utilizando o composto de métodos sugeridos: "***Draw Distance* definido, *Dithering* implementado e DoF aplicado**".

Figura 20 - Cena criada para o experimento, usando o composto de métodos sugerido

Foram criadas cenas com um número grande de instâncias dos objetos, para fazer *stress tests* (provas de esforço) e verificar com clareza a eficiência dos métodos. Algumas das métricas serão o número de *draw calls*(Chamadas de desenho na GPU), a taxa de quadros por segundo, e uma pesquisa de opinião sobre a qualidade visual das cenas. O mesmo ângulo específico com a câmera em movimento de Seno foi utilizado para fazer comparações entre:

- A visão com a otimização (*Draw Distance*) mas sem técnicas de redução de impacto na qualidade visual
- A visão com a otimização (*Draw Distance*) e com uma técnica de redução de impacto na qualidade visual (*Dithering*)
- A visão com a otimização (*Draw Distance*) e com duas técnicas de redução de impacto na qualidade visual (*Dithering* e *Depth of Field*)

7. Resultados

A partir do experimento realizado, usando como material a cena demonstrada no capítulo anterior, foram obtidos resultados expressivos sobre o uso do *Distance Culling* para otimização de performance, e do efeito *Dithering***[4]** para redução do impacto visual, também com *Depth of Field*. Todas as experimentações foram feitas com as mesmas condições de parâmetros de configuração da UE4. Na **Tabela 4** são demonstrados os resultados técnicos obtidos em cada configuração de uso das técnicas definidas neste trabalho.

	Taxa de Quadros	**Primitivas Estáticas (Draw Calls)**
Original sem *Draw Distance*	51fps a 53fps	1280 a 1510 média: meio termo
***Draw Distance* definido**	56fps a 63fps	820 a 970 média: meio termo
***Draw Distance* definido e *Dithering* implementado**	55fps a 62fps	820 a 970 média: alta
***Draw Distance* definido, *Dithering* implementado e DoF aplicado**	54fps a 61fps	820 a 970 média: alta

Tabela 4 - Comparação entre os métodos utilizados na mesma cena

Na **Figura 21** podemos ver a cena mais completa com os métodos implementados, "***Draw Distance* definido, *Dithering* implementado e DoF aplicado**", no momento em que a câmera está mais longe e mais próxima dos objetos. Temos também gráficos de desempenho em cada visão, mostrando em verde o tempo de renderização dos frames. Tempo de frame é o inverso de Taxa de quadros, por isso quanto menor o valor, melhor. Como descrito na tabela, quando estamos mais longe dos objetos, portanto tendo eles escondidos, obtemos uma taxa de 61 quadros por segundo. Quando estamos mais perto dos objetos, portanto tendo vários deles renderizados, este valor cai para até 54 quadros por segundo. Sendo assim, no pior caso desta composição de métodos, a performance ainda fica melhor que o melhor caso original, de 53 quadros por segundo.

Figura 21 - Cena do experimento, com Culling, Dithering e DoF, demonstrando gráfico de tempo de renderização de frame. Mais longe na esquerda, mais perto na direita.

As diferenças nas médias de Primitivas Estáticas vistas na **tabela 4** se dão pelo fato do *Dithering***[4]** demorar mais que um quadro para esconder primitivas, pois processo de transição leva em média 0.3 segundos. Então além dos objetos normalmente renderizados, também temos os objetos em transição somados na conta de primitivas. Porém, podemos ver que o impacto disto (perda de 1fps mínimo e 1fps máximo) não é grande se comparado com o ganho visual ao ter o efeito implementado. Isso pode ser visto também na pesquisa de opinião feita sobre as três opções que usam os métodos.

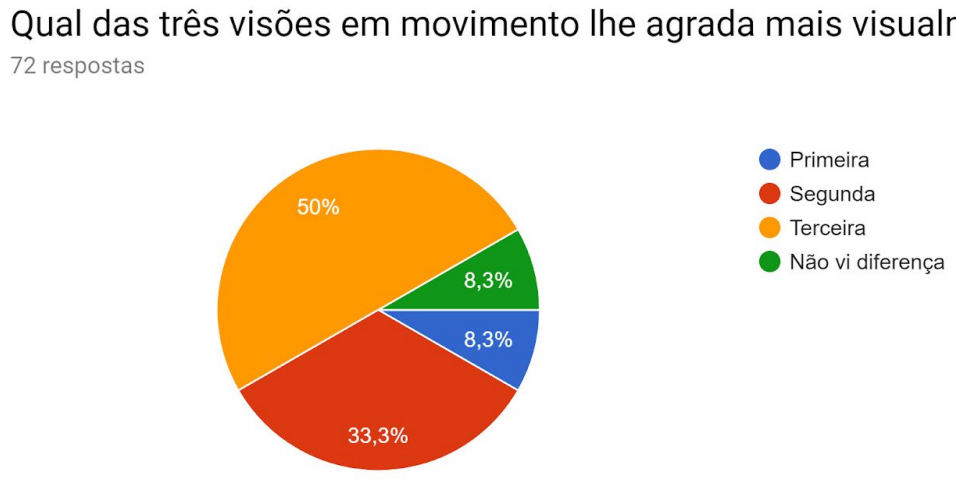

Figura 22 - Escolhas de preferência sobre as cenas na pesquisa de opinião

Para a pesquisa de opinião enviada na internet com respostas de escolha sobre os métodos na **Figura 22**, foram oferecidos 3 vídeos em loop mostrando as composições de métodos:

- Primeira: ***Draw Distance*** **definido**
- Segunda: ***Draw Distance*** **definido e** ***Dithering*** **implementado**
- Terceira: ***Draw Distance*** **definido,** ***Dithering*** **implementado e DoF aplicado**

Os vídeos estão disponíveis para serem vistos online**[22]**. Sem informações prévias sobre as diferenças entre os três vídeos, os pesquisados puderam escolher entre uma das três como a que mais agrada visualmente, ou então dizer se não tiver percebido diferença. Das 72 pessoas que responderam a pesquisa, 36 - exatamente metade - preferem "***Draw Distance*** **definido,** ***Dithering*** **implementado e DoF aplicado**", ou seja, o nosso composto de métodos sugerido, ainda com DoF. Ainda, 24 pessoas escolheram "***Draw Distance*** **definido e** ***Dithering*** **implementado**", também nossos principais métodos propostos de otimização e redução de impactos. Ou seja, do total, 60 das 72 pessoas (83.33%) percebe *Dithering* como um bom aparato visual.

Além disso, foi deixado como opcional responder o motivo da escolha, e vários dos entrevistados deram motivos como "Naturalidade" e "Fluidez" para a escolha do segundo e principalmente do terceiro, além de outras opiniões interessantes. Todas as opiniões colhidas, relacionadas com a preferência de cada um estão descritas no **Apêndice A** deste trabalho.

	Taxa de Quadros	**Preferências em pesquisa**
Draw Distance **definido**	56fps a 63fps	8.33%
Draw Distance **definido e** ***Dithering*** **implementado**	55fps a 62fps	33.33%
Draw Distance **definido,** ***Dithering*** **implementado e DoF aplicado**	54fps a 61fps	50%

Tabela 5 - Relacionando performance com preferências de estilo visual

Tomando a **Tabela 5** como referência vemos que é uma boa escolha sacrificar um pouco da taxa de quadros para ter uma maior qualidade visual usando estes métodos. Por estarmos tratando de plataformas menos potentes, é esperado que decisões

para uso de métodos de otimização como diminuição de *Draw Distance* estejam presentes para atingirmos performance aceitável pelo público. Dada a relação de custo-benefício entre perder um pouco desse ganho de performance para ter uma experiência mais agradável, é reafirmada a sugestão em utilizar os métodos de otimização e suas técnicas de redução de impacto propostas neste trabalho.

As decisões irão variar para cada projeto, como por exemplo nem todo estilo visual de jogo cabe a utilização de DoF, por exemplo. Então é necessário um estudo de caso específico para decidir quais se utilizar.

8. Conclusão

8.1. Contribuições

Tendo sido propostas diversas maneiras de otimizar jogos evitando o impacto negativo na qualidade visual, conclui-se que é de grande valor utilizar das análises e sugestões que constam neste trabalho. Vê-se uma oportunidade grande deste trabalho principalmente dado o crescimento acelerado de jogos e aplicações 3D em plataformas móveis ou híbridas. Muitas empresas encontram problemas na hora de migrar ou adicionar públicos para aproveitar esse crescimento, justamente por não terem conhecimento do que fazer quando seu jogo já foi otimizado ao máximo da maneira que era pensado. Para estas empresas, é considerado extremamente útil uma análise de oportunidades em diminuir drasticamente algumas propriedades antes não consideradas, para utilizar de ideias que minimizem esses impactos, por isso a importância deste trabalho para este mercado. O resultado do experimento também mostra que o principal método proposto é de grande valor para ajudar na solução do problema apresentado.

8.2. Limitações

A intenção deste trabalho era fazer não só uma análise de qualidade e validade dos métodos propostos, mas também uma análise comparativa entre eles, e descobrir quais são os mais efetivos tanto em performance quanto em compensação visual, porém pela limitação de tempo não foi possível fazer uma comparação extensiva, mas no lugar foram feitos também estudos de caso com jogos recém-lançados no mercado que mostram o uso ou não de alguns dos métodos propostos e quais resultados obtiveram com estas decisões.

Alguns dos métodos podem ser parcialmente dependentes de plataformas, Engines e outros fatores, então não necessariamente irão funcionar em alguns casos sem um esforço maior, portanto são um guia geral de como procurar por oportunidades de otimização sem impacto grande no visual.

Alguns dos métodos necessitam conhecimentos profundos de gráficos 3D e suas implementações para serem aplicados, principalmente se a ferramenta ou Engine utilizada não tem suporte nativo algum ao que se é pretendido.

Tópicos relacionados a modificação de código nativo de Engines não possuem muita documentação disponível na internet, e portanto podem ser mais difíceis de serem implementados.

8.3. Trabalhos Futuros

Além de realizar uma análise comparativa entre os métodos como especificado na seção de limitações, também se vê a oportunidade de criar um guia informativo em formato mais expressivo e visual para explicar os conceitos e compartilhar com desenvolvedores e empresas que possam estar passando pela necessidade em que este trabalho deseja ajudar.

Esta possível análise comparativa também inclui uma então pesquisa de opinião mais completa e de larga escala, para ter uma noção maior de opinião sobre os métodos propostos.

Alguns dos métodos propostos podem ser implementados e redistribuídos em forma de plugin para Engines conhecidas como UE4 e Unity3D.

Ao fim, pode ser proposto um método geral que, ao ter implementado vários dos métodos sugeridos, sirva como um gerenciador, que decida quais métodos devem ser ativados e desativados dinamicamente de acordo com o estado do jogo. Isto pode ser de grande vantagem aos jogos que precisam bastante de gerenciamento de recursos e performance para se manter em níveis estáveis e aceitáveis de execução e qualidade visual.

Referências

[1] Assarsson, U., & Möller, T. (1999). Optimized view frustum culling algorithms. *Chalmers University of Technology, Sweden*.

[2] Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., & Huebner, R. (2003). *Level of detail for 3D graphics*. Morgan Kaufmann.

[3] Ropinski, T., Wachenfeld, S., & Hinrichs, K. (2004). Virtual reflections for augmented reality environments. In *Int. Conference on Artificial Reality and Telexistence* (pp. 311-318).

[4] PIAZZA, Thomas A. et al. Method and apparatus for texture level of detail dithering. U.S. Patent n. 6,191,793, 20 fev. 2001.

[5] Demers, J. (2004). Depth of field: A survey of techniques. *Gpu Gems*, *1*(375), U390.

[6] Christiansen, K. R. The use of Imposters in Interactive 3D Graphics Systems. *Department of Mathematicsand Computing Science Rijksuniversiteit Groningen Blauwborgje*, *3*.

[7] Chehimi, F., Coulton, P., & Edwards, R. (2005, July). Evolution of 3D games on mobile phones. In *Mobile Business, 2005. ICMB 2005. International Conference on* (pp. 173-179). IEEE.

[8] Rime, 2017 - <http://www.tequilaworks.com/en/projects/rime/>

[9] Hosseini, M., Peters, J., & Shirmohammadi, S. (2013, February). Energy-budget-compliant adaptive 3D texture streaming in mobile games. In *Proceedings of the 4th ACM Multimedia Systems Conference* (pp. 1-11). ACM.

[10] Hwang, C., Pushp, S., Koh, C., Yoon, J., Liu, Y., Choi, S., & Song, J. (2017, October). RAVEN: Perception-aware Optimization of Power Consumption for Mobile Games. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking* (pp. 422-434). ACM.

[11] Jan KrassNigg, University of Aachen, Germany, Decals, Game Engine Gems, Volume One.. Cap. 20, Page 271.

[12] Koskela, T., & Vatjus-Anttila, J. (2015). Optimization techniques for 3D graphics deployment on mobile devices. *3D Research*, *6*(1), 8.

[13] Dynamic Resolution Rendering Article by doug-binks, published on July 13, 2011 - <https://software.intel.com/en-us/articles/dynamic-resolution-rendering-article>

[14] Dynamic 2D Imposters: A Simple, Efficient DirectX 9 Implementation, by Ashley Davis, published on Gamasutra, January 5, 2006

[15] Precomputed Visibility, Documentação da Epic Games Inc., <https://api.unrealengine.com/udk/Three/PrecomputedVisibility.html>

[16] Teach Like a Gamer: Adapting the Instructional Design of Digital Role-Playing Games, McFarland, 25 de maio de 2018, Cap. 3, Pág 65.

[17] Guia no fórum da plataforma Steam sobre *Precomputed Visibility*[15] <https://steamcommunity.com/sharedfiles/filedetails/?id=805273408>

[18] Doom, 2016 - <https://bethesda.net/en/game/doom-2016>

[19] Fortnite Battle Royale, 2017 - <https://www.epicgames.com/fortnite>

[20] Metacritic, <https://www.metacritic.com/>

[21] Fortnite has 78.3 million monthly players, according to Epic - <https://www.polygon.com/fortnite/2018/9/20/17884036/how-many-fortnite-monthly-players-2018>

[22] Vídeos para pesquisa de opinião: Primeiro - <https://streamable.com/py31s>, Segundo - <https://streamable.com/bw4pv>, Terceiro - <https://streamable.com/6jdw1>

[23] Wloka, M. M., & Zeleznik, R. C. (1996). Interactive real-time motion blur. *The Visual Computer*, *12*(6), 283-295.

Apêndice A

Respostas à pesquisa de opinião do experimento.

Não vi diferença	
Não vi diferença	Não consegui identificar nenhuma diferença!!
Não vi diferença	
Não vi diferença	Não consegui distinguir, mas aparentemente a 3 tem um Point of View maior, que deixa mais confortável
Não vi diferença	
Não vi diferença	
Primeira	
Primeira	
Primeira	
Primeira	Na primeira os objetos aparecem por completo. Na segunda, há um efeito pontilhado incômodo no aparecimento. Na terceira, o que me incomoda é o efeito borrado/falta de nitidez de alguns elementos na foto como os objetos que aparecem e parte do cenário ao fundo.
Primeira	
Primeira	
Segunda	foi só a qualidade mesmo
Segunda	O efeito de blur não me agrada
Segunda	A Primeira chama muita atenção ao objetos serem carregados totalmente na sua frente, por esse motivo estive tentando me decidir entre as outras duas , optei pela segunda pois na terceira por mais que seja suave os objetos sendo carregados , o "borrão" na profundidade me incomoda um pouco , tive a sensação que estava precisando usar óculos para focalizar algo ao fundo do mundo. Então por eliminação escolhi a segunda opção.
Segunda	Melhor conforto visual e noção de profundidade.
Segunda	É a mais suave. A primeira simplesmente "pops into existence" e a segunda tem muito blur
Segunda	
Segunda	A segunda possui uma transição mais suave. A terceira também é suave, mas o blur incomoda a vista.
Segunda	
Segunda	
Segunda	Escolhi essa opção pois achei que a forma como os objetos que somem e aparecem, quando a câmera se afasta ou se aproxima, surgem é mais agradável aos olhos. Na 1a opção, os objetos simplesmente surgem imediatamente, o que achei "invasivo" e, apesar que a 3a também começa com o efeito meio "fade in/out" que tem na 2a, ele interrompe o fade no final em favor de um "pop in/ou", o que ainda achei invasivo. Assim, o fade in/out exibido na 2a opção não é tão chamativo quanto os outros, o que achei mais agradável.
Segunda	
Segunda	Perspectiva de aproximação, se for pra um game, confesso que não prefiro nenhuma das três.
Segunda	
Segunda	
Segunda	
Segunda	O padrão me lembra como se estivessem surgindo por detrás de folhagens, me parece mais orgânico.
Segunda	
Segunda	A primeira se aproxima muito rapidamente, é como se eu estivesse em um balanço. A terceira demora muito para aproximar e parece que tem mais janelas sendo criadas no ar. A segunda foi a que achei mais harmonica.
Segunda	Existe uma suavidade na relação distância e aparecimento/desaparecimento das janelas. Torna o fluxo natural.

Figura 23 - Primeira parte do Apêndice A

Segunda	
Segunda	na segunda as "peças" demoram um pouco mais pra se encaixar e com isso fica mais agradável a visão
Segunda	
Segunda	
Segunda	Não vi muita diferença, mas a segunda parecia que fluia melhor
Terceira	pq parece mais desfocada, ajuda a esconder
Terceira	a transição é mais suave
Terceira	A noção de profundidade do ambiente na terceira é mais enxuta e a transição mais suave.
Terceira	
Terceira	A terceira foi mais "smooth". Teve até impressão que teve mais dessas janelas na tela, por mais que são a mesma quantidade. Tive menos sentimento de abandono. A primeira achei muito ríspida. Já a segunda melhorou bastante e nem reclamaria. Já a terceira estou aqui elogiando.
Terceira	a imagem parece mais limpa e agradavel. O zoom da uma perspectiva mais legal.
Terceira	Efeito mais gradativo, menos brusco.
Terceira	
Terceira	Achei mais natural
Terceira	Porque os elementos que estão mais longes também estão desfocados
Terceira	Me deu uma sensação mais nítida de começo, meio e fim, com as janelas (?!) entrando um pouco depois do céu limpo
Terceira	O movimento mais lento me agrada mais aos olhos. Não sei o porque, so parece mais suave.
Terceira	Não faço a menor ideia. O pressuposto de que você é obrigado a escolher uma dentre muitas opções faz com que você comece pelo processo de eliminação das que lhe são visualmente mais desagradáveis (eliminei a primeira, de cara). Ficando entre a segunda e a terceira, penso a última ser mais agradável visualmente. O efeito de transição dá um ar de naturalidade maior do que o efeito da segunda. É isto.
Terceira	A terceira transição é a que me pareceu mais suave, e isso me fez entender melhor o que estava acontecendo. A segunda transição foi a segunda melhor, mas não mudou muito.
Terceira	Graças ao cenário que parece ser meio misterioso eu gostei mais da terceira que deu um ar de nevoeiro mais forte.
Terceira	da mais impressão que estamos nos afastando/nos aproximando dos objetos
Terceira	Pra enxergar o ambiente de forma mais paronamica
Terceira	
Terceira	É muito mais suave e fluida do que as outras duas. O desfoque nas imagens mais a fundo também ajudam bastante a manter o foco na imagem mais à frente durante a aproximação.
Terceira	Fluidez
Terceira	A segunda e a terceira tem suavidade de visibilidade de objetos. A terceira n tem flicker do mar no lado direito.
Terceira	Me pareceu mais lenta, por algum motivo me fez observar o cenário com mais atenção. E as mini-lápides desaparecendo eram relaxantes, parecia um ASMR sem som
Terceira	A transição parece ser mais suave do que as outras
Terceira	
Terceira	Parece mais natural
Terceira	mais fluido e natural q os demais
Terceira	a terceira é interessante porque as coisas aparecem desfocadas antes, parece que to me aproximando "mais real"
Terceira	A visão começa aberta e é como se eu tivesse caminhando pra frente, isso me agrada. Caminhar/explorar como se eu tivesse andando e já sabendo mais ou menos a visão geral.
Terceira	A primeira - Parece defeito as coisas aparecendo e desaparecendo abruptamente. A segunda - parece limitações computacionais que fizeram aquilo. A terceira - parece que foi feita para ser daquele jeito fluido.
Terceira	Achei o movimento mais smooth dentre as 3 e com o ceu se abrindo mais quando imagem esta longe.
Terceira	
Terceira	Fluidez
Terceira	visao mais ampla do cenário
Terceira	A transição do primeiro é muito "dura", dá a impressão que é algum glitch. Um passo pra frente aparece, um para trás e pisca de volta. O segundo e o terceiro são agradáveis no meu ponto de vista, acho que são mais questão de estilo. Em ambos dá pra perceber que é questão de campo de visão, e não de qualquer problema de gráfico.
Terceira	As animações são mais suaves.
Terceira	ela é mais fluida que as outras duas

Figura 24 - Segunda parte do Apêndice A