Rodrigo Araújo Borges

Renderização Fotorrealista de Borrões de Movimento para Realidade Aumentada

Recife

2019

Rodrigo Araújo Borges

Renderização Fotorrealista de Borrões de Movimento para Realidade Aumentada

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Computação

Curso de Bacharelado em Ciência da Computação

Orientador: João Paulo Lima

Recife

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

B732r Borges, Rodrigo Araújo
Renderização fotorrealista de borrões de movimento para realidade aumentada / Rodrigo Araújo Borges. - 2019.
54 f. : il.

Orientador: João Paulo Silva do Monte Lima.
Inclui referências.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Ciência da Computação, Recife, 2019.

1. Realidade aumentada. 2. Computação gráfica. 3. Fotorrealismo. 4. Borrão de movimento. I. Lima, João Paulo Silva do Monte, orient. II. Título

CDD 004

MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO
UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO (UFRPE)
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

http://www.bcc.ufrpe.br

Ficha de Aprovação do Trabalho de Conclusão de Curso

Trabalho defendido por **RODRIGO ARAÚJO BORGES** às 16:00 do dia 05 de dezembro de 2019, no Laboratório 35 do Departamento de Computação, como requisito para conclusão do curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, intitulado " **Renderização Fotorrealista de Borrões de Movimento para Realidade Aumentada**", orientado por João Paulo Silva do Monte Lima e aprovado pela seguinte banca examinadora:

João Paulo Silva do Monte Lima
DC/UFRPE

Valmir Macário Filho
DC/UFRPE

Agradecimentos

Agradeço primeiramente aos meus pais, Paloma e Sergio, por me proporcionarem com belos exemplos de educação e uma vida qual eu não poderia desejar melhor. Mesmo optando por uma carreira que me leve a passar mais tempo em frente a um computador pude contar com o apoio deles.

A minha irmã, Raíssa, por todas as ajudas em várias etapas da vida e de meu aprendizado. Aos meus avós, tios e primos por todo o apoio até agora.

Agradeço também ao professor João Paulo Lima pela orientação e ensinamentos durante este trabalho e suas disciplinas.

Aos meus amigos e colegas de curso que me fizeram companhia nesses últimos anos, especialmente aqueles que tornavam as viagens de ida e volta mais prazerosas.

Por fim, agradeço a todos os professores e profissionais da Universidade Federal Rural de Pernambuco pelo ambiente acolhedor, repleto de sabedoria e propício ao crescimento.

“O ontem é história, o amanhã é um mistério, mas o hoje é uma dádiva. Por isso chama-se presente.”

(KUNG... 2008)

Resumo

A tecnologia de Realidade Aumentada permite a inserção de elementos virtuais em cenários reais visualizados em tempo real por meio de algum dispositivo. Para que isso seja possível é realizado um rastreamento das características importantes do ambiente real, seja por meio da câmera de um smartphone ou outros tipos de sensores. Um dos principais focos encontrados em aplicações de Realidade Aumentada é a imersão do usuário nessa realidade, isto é, fazê-lo acreditar que os elementos artificiais visualizados existem e estão presentes no mesmo ambiente que ele.

Existem uma variedade de técnicas para ampliar a imersão do usuário num ambiente de Realidade Aumentada, sendo uma das mais comuns a renderização de elementos virtuais com gráficos que aproximam sua aparência a de elementos do mundo real. Tanto na Computação Gráfica como na Arte esse estilo é conhecido como Fotorrealismo. Na renderização fotorrealista são considerados aspectos como sombra, refração da luz e oclusões, além de outros perceptíveis em um ambiente real. Fora esses aspectos existem também efeitos causados pelo meio de captura que realiza o rastreio. No caso das câmeras digitais, um efeito recorrente é o registro de borrões em momentos em que um elemento está se movendo. Para uma experiência de Realidade Aumentada imersiva com elementos fotorrealistas é importante que os objetos virtuais sofram desses efeitos assim como os elementos reais de maneira constante, satisfazendo a demanda em tempo real.

Este trabalho tem como objetivo desenvolver uma solução eficiente para renderização de borrões de movimento fotorrealistas em Realidade Aumentada. Implementar a solução no motor gráfico Unity permite futuramente que outros projetos de Realidade Aumentada desenvolvidos nesse mesmo motor possam usufruir de seus benefícios. Foi feito um estudo acerca dos desenvolvimentos existentes na linha de Realidade Aumentada Fotorrealista, e entre os quais desenvolveram alternativas para a renderização de borrões de movimento, foi escolhido aquele que apresentava resultados mais satisfatórios como base para os algoritmos desse trabalho. Os resultados obtidos foram avaliados visualmente e por comparações entre a frequência de quadros por segundo exibidos.

Palavras-chave: Realidade Aumentada, Computação Gráfica, Fotorrealismo, Borrão de Movimento.

Abstract

Augmented Reality technology allow us to insert virtual elements in real environments viewed in real time through a device of some sort. To make it possible a tracking of important features in the environment is performed, be it by the use of a smartphone camera or other types of sensors. One of the main focus found in Augmented Reality Applications is providing immersion of the user in that reality, that is, by making they believe the artificial elements being viewed are present in the same place as they are.

There is a variety of techniques for improving the user immersion in a Augmented Reality environmnent, being one of the most common ones the rendering of virtual elements with graphical appearence closer to that of a real world element. In both Computer Graphics and Art this style is known as Photorealism. In photorealistic rendering some aspects are taken into consideration, such as shadows, light refraction and occlusion, among others noticeable in the real world. Other than these there are also effects caused by the means of capture performing the tracking. In the case of digital cameras, a recurring effect is the appearance of blur in moments where an element is moving. To provide an immersive Augmented Reality experience with photorealistic elements it is important that the virtual objects undergo these effects just as the real elements do constantly, satisfying the real time demand.

This work aims to develop a efficient solution for the rendering of photorealistic motion blur in Augmented Reality. Implementing the solution in the game engine Unity allows for other future Augmented Reality projects developed in this same engine to benefit from its features. A study was made about existing developments in the area of Photorrealistic Augmented Reality, and among the ones that offered alternatives for the rendering of motion blur, the one with the best shown results was chosen as the basis for the algorithms in this work. The obtained results were validated visually and through comparisons between the frequency of frames per second displayed.

Keywords: Augmented Reality, Computer Graphics, Photorealism, Motion Blur.

Lista de ilustrações

Sumário

1 Introdução

A projeção de elementos virtuais em cenários reais já obteve grandes desenvolvimentos na área de computação gráfica. Com os resultados desses avanços facilmente perceptíveis na indústria do cinema, em filmes como Vingadores: Ultimato, ambientes e personagens inteiros chegam a ser renderizados em harmonia com os atores, gerando situações onde o espectador não sabe o que é ou não real (OBSERVATóRIO DO CINEMA, 2019). A produção de resultados com tal fidelidade em aplicações de execução em tempo real, como Realidade Aumentada (RA) e Realidade Mista (RM), propõem um desafio bem maior.

Os avanços e o acesso a essas tecnologias estão cada vez maiores. Em 2016 houve cerca de 20 milhões de pessoas jogando Pokémon GO em seus dispositivos móveis com o auxílio de RA. Recentemente, em maio de 2019, foi anunciado o jogo Minecraft Earth que propõe uma interação em RM entre jogadores e elementos virtuais posicionados no mundo real (THE VERGE, 2019).

Para ter uma experiência fidedigna em algumas aplicações de RM e RA torna-se necessário que os objetos sintéticos se misturem com os objetos reais do ambiente, sendo requisitada a renderização dinâmica de características como interações luminosas e efeitos consequentes do meio de captura. Muitas dessas aplicações de renderização acabam tendo que simplificar aspectos de seu processo para satisfazer a demanda em tempo real, por exemplo o sombreamento, prejudicando o realismo pela falta de interação entre a iluminação sintética e a iluminação real (FRANKE, 2014). A simulação dessas características a serem replicadas nos elementos virtuais também deve levar em conta as limitações da plataforma alvo e do meio de captura. Para a geração de efeitos de iluminação fiéis é necessária a captura das condições de iluminação do ambiente real com precisão, sendo essa uma limitação a ser superada considerando o campo de visão restrito de um dispositivo (Rohme, K. et al., 2014). Além disso temos as falhas visuais derivadas da qualidade de captura da câmera utilizada, causando efeitos como ruído, distorções e borrões de movimento. Esse último um dos maiores obstáculos de reprodução na RA e RM pois, além da necessidade de replicar o efeito de borrão na renderização, também atrapalha o funcionamento de algoritmos de rastreamento (Park, Y.; Lepetit, V.; Woo, W., 2012). Borrões de movimento são causados quando a câmera ou um objeto observado se movem durante o momento em que o obturador do dispositivo abre e fecha. É durante esse intervalo que o dispositivo de carga acoplada, responsável pela captura de luz a ser registrada nos pixels da imagem, é exposto (Mei, C.; Reid, I., 2008).

Existem trabalhos que apresentam resultados visualmente convincentes de borrões de movimentação em elementos virtuais, inseridos num ambiente real, que serão apresentados mais adiante. Neste trabalho será apresentada uma solução para a renderização do efeito de borrão de movimento no motor gráfico Unity, utilizando do pacote de desenvolvimento de software para RA Vuforia.

1.1 Problema de Pesquisa

Um dos fatores mais importantes ao misturar o real com o virtual é a coerência visual. Tal propriedade favorece a imersão do usuário ao contribuir com a noção de relação espacial, radiométrica e geométrica entre os elementos sintéticos e reais. Objetos virtuais que apresentem interações realistas com a luz real são de grande interesse para aplicações em áreas como entretenimento, medicina e educação (Kán, P.; Kaufmann, H., 2013). No trajeto para uma experiência cada vez mais fotorrealista em RA, a replicação em objetos virtuais das interações entre a luz e o meio de captura são fundamentais na coerência visual. A falta dessas interações ressalta a diferença entre os elementos originados da realidade física e os elementos gerados na virtualidade, podendo ser considerada uma "lacuna da realidade" (Fischer, J.; Bartz, D.; Straßer, W., 2006). Um dos efeitos produzidos pela câmera no processo de captura que pode ser replicado nos elementos virtuais da RA, incrementando a coerência visual, é o borrão de movimento.

Considerando desenvolvimentos existentes na área de RA em diversas plataformas envolvendo renderização e rastreamento, há uma busca notável pela imersão através do fotorrealismo. Para aproximar-se cada vez mais dessa experiência é preciso adaptar a renderização dos elementos virtuais a consequências consideradas como imperfeições do meio de captura. Sendo o borrão de movimentação uma dessas imperfeições, como podemos produzir esse efeito com fidelidade para RA?

1.2 Objetivos

Considerando o problema de pesquisa citado na seção anterior, encontram-se a seguir os objetivos estabelecidos para esse trabalho.

1.2.1 Objetivo Geral

Elaborar uma solução para a ausência de efeitos de borrões de movimento em RA no motor gráfico Unity em conjunto com o kit de desenvolvimento de software Vuforia.

1.2.2 Objetivos Específicos

Tendo em vista o objetivo geral definido na subseção antecedente, este trabalho visa cumprir os seguintes objetivos específicos:

- Desenvolver uma solução para a renderização de um objeto virtual com borrões de movimento em um ambiente de RA.
- Parametrizar a renderização dos efeitos de borrão de movimento de acordo com o movimento relativo entre a câmera e o alvo de forma coerente.

1.3 Metodologia

O presente trabalho iniciou-se por volta de abril de 2019, após a proposta do orientador Dr. João Paulo Lima para um projeto no âmbito de RA. O mesmo indicou o estudo de trabalhos dos quais entre eles estavam (FRANKE, 2014), (FRANKE, 2013). Em (FRANKE, 2017), o autor especifica o desenvolvimento de efeitos de pós processamento para replicação das falhas reproduzidas por câmeras menos potentes, e como estes são potenciais melhorias para seus trabalhos. Examinando essas falhas foi definida como foco do projeto a replicação de borrões de movimento em RA.

Foi realizado um estudo bibliográfico a respeito do foco definido acima envolvendo conceitos de RA, dispositivos de captura, fotorrealismo, rastreamento e algoritmos para renderização de efeitos de pós processamento. Nessa pesquisa não foi encontrado trabalho que apresentasse uma solução que implementasse os efeitos de borrões de movimento para RA no Unity, um dos motores gráficos mais populares no desenvolvimento de jogos (TOFTEDAHL; ENGSTRÖM, 2019).

1.4 Estrutura do Trabalho

O restante deste trabalho estrutura-se inicialmente com o segundo capítulo contendo o levantamento dos trabalhos relacionados como forma de justificativa para a abordagem proposta. O terceiro capítulo inclui um aprofundamento nos conceitos fundamentais ao trabalho, sendo dividido nos focos de RA (Seção 3.1) e Fotorrealismo (Seção 3.2). O capítulo 4 apresenta a solução proposta e seu processo de desenvolvimento, detalhando as ferramentas e técnicas utilizadas e os resultados obtidos. Por fim temos o capítulo 5 com a conclusão e propostas para trabalhos futuros.

2 Trabalhos Relacionados

Como podemos verificar em (KRONANDER et al., 2015), já existem diversos métodos consistentes para renderização de objetos virtuais em cenários reais com interação luminosa entre ambos. Pra alcançar resultados tão satisfatórios em RA, de maneira que o real e o virtual se misturem convincentemente em tempo real, além do problema da falta de interação luminosa há a abstinência dos efeitos causados pela câmera. Para resolver isso existem duas opções: remover as imperfeições e distorções do meio de captura ou adaptar o elemento virtual a essas características, sendo essa segunda a opção mais fácil (KLEIN; MURRAY, 2010).

A ferramenta de testes de RM Dirtchamber, desenvolvida por T. Franke em (FRANKE, 2014) e (FRANKE, 2013), conta com duas técnicas para a aproximação de um ambiente com interações luminosas fotorrealistas. Essas técnicas são o Delta Voxel Cone Tracing (DVCT) e Delta Light Propagation Volume (DLPV). No DVCT o cenário e o objeto virtual passam pelo processo de voxelização, onde são projetados dentro de um volume cubicular composto por voxels. Esses voxels são como pixels tridimensionais, que ao invés de ocupar um quadrado numa área ocupam um cubo em um volume. Essa solução reduz o custo computacional ao diminuir a complexidade geométrica do ambiente e permite utilizar os voxels para armazenar informações de como a região preenchida por estes deverá se comportar no ambiente de RM. O DVCT provê a renderização de um ambiente com efeitos de iluminação indireta, incidida pela interação do virtual com o real, incluindo sangramento (*Color Bleeding*), sombras e reflexos. Embora o DVCT trabalhe com espelhos para a geração de reflexo, outros efeitos especulares como os obtidos em RA por P. Kán e H. Kaufmann em (KAN; KAUFMANN, 2012) e (KAN; KAUFMANN, 2013), como refração e cáustica, não são apresentados. Enquanto o DLPV tambem lida com interações luminosas em RM, diferentemente do DVCT, ele trabalha com pontos em um volume tridimensional e não produz efeitos de reflexos em seus resultados. O DVCT, a técnica mais recente do Dirtchamber, é conhecida por ser a primeira solução a apresentar reflexos gerados em tempo real para superfícies foscas (FRANKE, 2014), como na figura 1.

Outro trabalho mais recente que também envolve o uso de voxelização é o de Regenbrecht et al. em (REGENBRECHT et al., 2017). Os autores utilizam o conceito de Realidade Mediada por Computador(*Computer-Mediated reality*) com foco no desenvolvimento da sensação de presença dos elementos virtuais. Nessa realidade são incluídas técnicas de Realidade Virtual (RV) e RM, onde o usuário usufrui de óculos de RV para visualizar o ambiente ao seu redor com a inclusão de elementos virtuais. Durante a execução, para a captura e voxelização da cena, foi utilizado o motor grá-

Figura 1 – Renderização Fotorrealista de reflexos em superfície fosca no Dirtchamber (FRANKE, 2014)

fico Unity e um sensor de profundidade Kinect posicionado em frente ao usuário, que através do óculos de RV o enxerga como um espelho. Assim como no DVCT, e similar a pixels em uma tela, o tamanho dos voxels influencia diretamente no fotorrealismo e no desempenho do ambiente. Voxels maiores produzem um cenário menos detalhado e com maior desempenho, enquanto voxels menores acarretam em um cenário mais detalhado ao custo de desempenho. Buscando o equilibrio entre esses dois fatores decidiram que seus voxels seriam compostos por lados de 0,8 cm. Centrados em deixar imperceptível as diferenças entre objetos reais e virtuais no ambiente simulado, esse trabalho acaba por diminuir o fotorrealismo dos objetos reais reais voxelizados para ficarem mais próximos da aparência dos objetos puramente virtuais. Embora as soluções de Regenbrecht et al. e de Franke entreguem resultados de RA e RM consistentes de um ponto de vista estático, ambas poderiam utilizar do efeito de borrão de movimento para ampliar sua fidelidade ao fotorrealismo na presença de movimento.

Em (OKUMURA; KANBARA; YOKOYA, 2006), os autores propõem um método de rastreamento de marcadores e renderização do objeto virtual desfocado ou com borrões de movimento para RA baseado na estimação da Função de Espalhamento Pontual (*Point Spread Function - PSF*). A PSF é uma função que descreve como ocorre o efeito de borrão de espalhamento de um ponto ao ser registrado. Ao detectar o marcador em movimento, o algoritmo utiliza da intensidade dos pixels ao redor das beiradas desse marcador para estimar os parâmetros da PSF. Após isso, é estimada a posição e orientação da câmera em relação ao marcador considerando os borrões registrados. Com essas informações coletadas é feita então a renderização do objeto com efeito de borrão baseado no PSF encontrado. Mesmo propondo uma alternativa para ambas as situações de desfoque e borrões de movimento no objeto virtual, essa solução depende de que o elemento virtual seja renderizado junto ao marcador para funcionar adequadamente.

Já em (Park, Y.; Lepetit, V.; Woo, W., 2012) os autores focam no rastreamento

de um objeto real para a geração de um virtual. Para esse rastreamento é utilizado um modelo de formação de imagens com variações do algoritmo ESM (*Efficient Second-order Minimization*). São geradas imagens do objeto a ser rastreado com diferentes intensidades de borrões. Ao registrar cenas do objeto em movimento que apresentem borrões, essas são comparadas com as imagens geradas para iniciar e manter o rastreio. Com base nesses borrões gerados é feita uma estimativa do movimento que está sendo realizado para a renderização do objeto sintético com borrão de movimento. Essa renderização é então realizada misturando sequências de imagens do elemento virtual. Para criar um efeito de borrão fotorrealista seriam necessárias dezenas de renderizações do objeto, que se feitas em 3D acarretariam em um grande custo para um sistema que deve ser executado em tempo real. Dessa forma, os autores optaram por renderizar uma pequena quantia de imagens do elemento virtual no trajeto do movimento e então gerar quadros intermediários ao distorcer essas imagens. Esses quadros são chamados de *Intraframes*, e como a distorção de uma imagem 2D é normalmente mais rápida que uma renderização 3D, esse método permite a geração de efeitos de borrões de movimento corretamente com a eficácia necessária para um processo em tempo real (Park, Y.; Lepetit, V.; Woo, W., 2012).

No trabalho de Mei e Reid, (Mei, C.; Reid, I., 2008), também são desenvolvidas soluções para o rastreamento na presença de borrões de movimento e a geração de borrões. Diferente do estudo de Park, Lepetit e Woo citado anteriormente, os borrões produzidos no trabalho de Mei e Reid são voltados para auxiliar no rastreio. Os autores também desenvolveram uma solução para o rastreio baseado na geração e comparação de imagens borradas. O algoritmo desenvolvido para a geração dos borrões utiliza uma técnica próxima a convolução de integral de linha, onde uma função é aplicada a todos os pixels da imagem, espalhando suas cores dependendo de suas coordenadas e criando o efeito de borrão.

Fischer, Bartz e Straßer trabalham em (Fischer, J.; Bartz, D.; Straßer, W., 2006) com 3 características entre as quais um objeto virtual renderizado deve apresentar para misturar-se com o ambiente real capturado. São essas: ruído, antiserrilhamento (*anti-aliasing*) e borrão de movimento. O ruído é um efeito causado pela câmera, facilmente perceptível em dispositivos mais antigos, onde os pixels aparentam constantemente estar trocando seu tom ou cor. O antiserrilhamento é um efeito que funciona como medida contra o efeito de serrilhamento. Perceptível nos elementos virtuais renderizados, o serrilhamento ocorre quando é possível enxergar a divisão entre os pixels que pertencem a tal elemento e os pixels vindos do ambiente real, criando essa impressão de serrilhado, que não deveria existir. Como esse trabalho trata do problema da ausência do borrão de movimento fotorrealista, não será aprofundado nas soluções oferecidas para ruído e serrilhamento. Para resolver esse problema é proposto utilizar os dados de posicionamento da câmera, calculando um vetor borrão (*blur vector*) baseado na

localização do objeto a ser renderizado em relação ao meio de captura entre 2 quadros consecutivos. Se o vetor borrão for maior que um limiar, geralmente de 5 a 10 pixels, é ativada a renderização do borrão de movimento. O borrão artificial é simulado ao misturar repetidamente a imagem do objeto virtual em posições ao longo do vetor borrão adicionando um efeito de transparência.

Klein e Murray também desenvolvem em (KLEIN; MURRAY, 2010) soluções para efeitos derivados do meio de captura, especificamente de pequenas câmeras. Utilizando dados previamente obtidos por especificações do dispositivo ou calibração, efeitos como ruído, distorção e aberração cromática são replicados no objeto virtual renderizado em RA. Cada imagem gerada é resultante de um processo separado em 10 partes, onde em cada uma é tratado um efeito diferente, sendo um deles o borrão de movimento. Somente o movimento rotacional da câmera é considerado na renderização do borrão de movimento. A intensidade do borrão é estimada utilizando um sistema de gradeamento da imagem, onde é considerado o movimento rotacional da imagem nessa grade em um intervalo de 33 milissegundos. Com a intensidade do borrão em cada célula da grade determinada, os pixels do objeto virtual pertencentes a essas células têm suas cores misturadas no sentido do movimento, dando a impressão de borrão. O borrão de movimento por translação e movimentação do objeto virtual não é considerado por ser considerado difícil, já que requer múltiplas renderizações geométricas.

As soluções encontradas para renderização de um objeto sintético em um cenário real com características fotorrealistas não levam em consideração a aplicação de borrões de movimento. Embora o pacote de desenvolvimento ARKit 3 para a versão mais recente do sistema operacional iOS, o iOS 13, possua suporte para a renderização de diversas características fotorrealistas, incluindo borrões de movimento, essa solução é de distribuição exclusiva para dispositivos fabricados pela Apple (DOVE, 2019). Enquanto isso, entre os resultados obtidos pelos trabalhos envolvendo borrões de movimento fotorrealistas em RA, não foram encontradas soluções para um motor gráfico multiplataforma popular como o Unity. Um possível motivo para a pequena quantidade de trabalhos com foco na renderização de borrões de movimento para RA seria o desenvolvimento contínuo de câmeras melhores, tornando cada vez mais imperceptível a ocorrência desse efeito.

3 Fundamentação Teórica

Neste capítulo encontra-se um breve aprofundamento dos principais conceitos utilizados neste trabalho.

3.1 Realidade Aumentada e Mista

Nos dias atuais, aplicações de RA podem ser facilmente encontradas nos serviços de distribuição digital de aplicativos, tais como Google Play (GOOGLE..., 2019) e App Store (APP..., 2019). Englobando diversos cenários, como fotografia, jogos digitais e arquitetura, essas aplicações usufruem dos desenvolvimentos nas áreas de computação gráfica e visão computacional para submeter o usuário a uma nova realidade.

3.1.1 Definição

O termo "realidade" é derivado da palavra "real". Segundo o dicionário Michaelis (EDITORA MELHORAMENTOS LTDA., 2015) real é definido como "que não é imaginário; que tem existência no mundo dos sentidos; concreto, objetivo". Uma aplicação de RA sobrepõe um cenário real com elementos virtuais de forma que aparente que ambos coexistem em um mesmo ambiente. Azuma et al. define que um sistema de RA segue as seguintes propriedades:

- Combina objetos reais e virtuais em um cenário real;
- Executado de forma interativa e em tempo real;
- Registra (alinha) objetos reais e virtuais uns com os outros.

Essa definição não está limitada ao equipamento utilizado e nem ao sentido humano da visão. RA pode também incluir os outros sentidos, como a audição. Um exemplo foi a exibição comemorativa de 30 anos da franquia Final Fantasy, que contou com um sistema de áudio de RA onde, com o auxílio de fones de ouvido e rastreadores, os usuários escutavam as trilhas sonoras e as vozes das personagens da franquia de acordo com a sessão que estivessem (SQUARE ENIX CO., LTD., 2017).

Virtualidade Aumentada (VA) e RV são diferentes de RA. As três fazem parte do contínuo Realidade-Virtualidade definido por P. Milgram e F. Kishino em (Milgram, P.; Kishino, F., 1994). Em um ambiente de RV todos os elementos são gerados por computador. Já num ambiente de VA o usuário navega por um mundo majoritariamente vir-

tual, podendo interagir com objetos sintéticos ou reais (Ternier, S. et al., 2012). Como podemos ver na Figura 2, RA e VA estão localizadas opostas uma a outra dentro do intervalo denominado Realidade Mista (RM). Fora desse intervalo temos o Ambiente Real de um lado e o Ambiente Virtual (mesmo que RV) do outro. Dada essa representação pode-se dizer que a realidade da RA é seu fator primário enquanto a virtualidade o secundário (Yeon Ma, J.; Soo Choi, J., 2007).

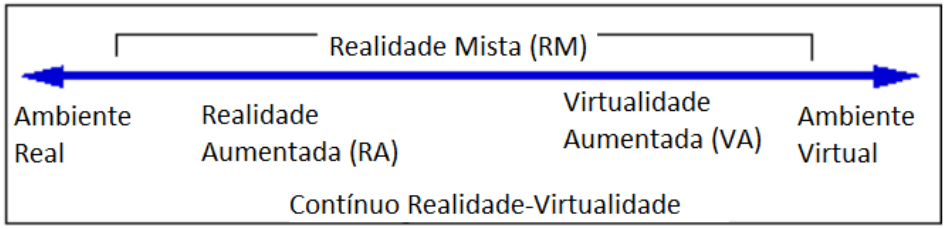

Figura 2 – Representação do Contínuo Realidade-Virtualidade.
(Milgram, P.; Kishino, F., 1994)

P. Milgram e F. Kishino classificam RM como toda parte entre o Ambiente Real e o Ambiente Virtual, englobando RA e VA. Com o passar dos anos outro conceito para RM surgiu. Como podemos ver em (INTEL, 2019), (ANIWAA, 2018) e (MARR, 2019), RM é definida como uma realidade própria, assim como RA ou VR, onde o usuário participa ativamente nesse ambiente interagindo com elementos reais e virtuais.

3.1.2 Dispositivos de Exibição

Existe uma variedade de dispositivos que funcionam como meios a aplicações de RA e RM. Em (Azuma, R. et al., 2001) esses dispositivos são classificados entre as seguintes 3 categorias:

3.1.2.1 Head-Worn Displays (HWD)

Também conhecido como Head-Mounted Displays (HMD), são os dispositivos acoplados a parte da cabeça do usuário. Estes podem conter uma lente transparente pelo qual pode-se observar os elementos reais com os elementos virtuais sendo projetados diretamente na lente, como o HoloLens (MICROSOFT, 2019), ou um visor digital onde o ambiente de RA é apresentado, como o Windows Mixed Reality (ACER, 2019). Nos HWDs de visores digitais o ambiente real é capturado por câmeras acopladas. Como podemos ver na Figura 3, mesmo um HWD que utilize lentes transparentes pode usufruir de meios de captura, tais como câmeras, para rastreio do ambiente real e interações do usuário. Há também HMDs de Virtual Retinal Display, onde as imagens virtuais são projetadas diretamente na retina do usuário, como o Magic Leap One (MAGIC LEAP, 2019).

Figura 3 – Usuário do HoloLens 2 da Microsoft interagindo com um elemento virtual com as mãos rastreadas pelas câmeras frontais.
(MICROSOFT, 2019)

3.1.2.2 Dispositivos de Projeção

Nesse sistema os elementos virtuais são projetados por cima dos elementos físicos, sem precisar de lentes ou telas posicionadas diretamente em frente aos olhos do usuário. Na Figura 4 é possível observar a *Therapeutic Lamp*, sistema de projeção de RA para tratamento de fobias de animais pequenos (Wrzesien, M. et al., 2013).

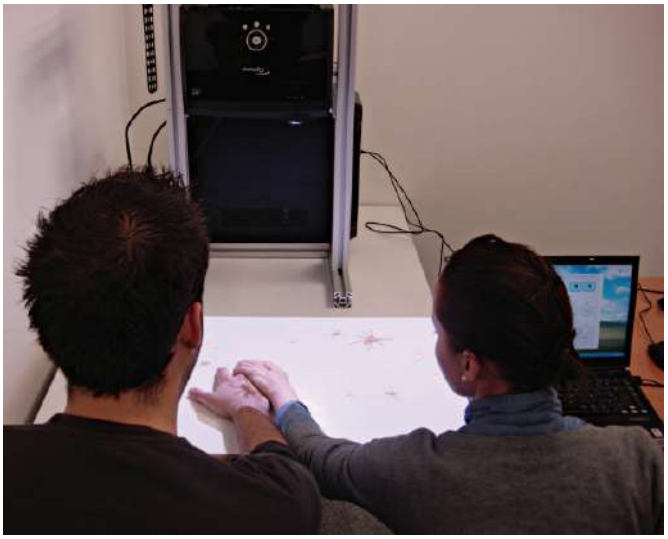

Figura 4 – Therapeutic Lamp. A terapeuta (direita) manipula os animais virtuais a serem projetados para o tratamento da fobia do paciente (esquerda).
(Wrzesien, M. et al., 2013)

3.1.2.3 Dispositivos portáteis

Dispositivos que podem ser facilmente transportados pelo usuário, como celulares, usufruindo de câmeras e um visor para criar uma experiência similar a uma "janela" para a RA. Na Figura 5 podemos observar o jogo Pokémon GO, que permite posicio-

nar o elemento virtual a ser renderizado em harmonia com a superfície do ambiente. Também pode-se fazer uso de uma câmera frontal para gerar um efeito de “espelho”, conceito utilizado em aplicações de RA com rastreamento facial.

Figura 5 – Pokémon GO para smartphones utiliza RA em sua mecânica de jogo. (STATT, 2017)

3.1.3 Rastreamento

Para que uma aplicação de RA posicione os elementos virtuais adequadamente é necessário que haja um rastreamento de informações relevantes do ambiente real. As tecnologias de rastreio utilizadas em RA possibilitam que a terceira propriedade definida por Azuma seja satisfeita de forma a aparentar que os objetos virtuais fazem parte do mundo físico (Billinghurst, M.; Clark, A.; Lee, G., 2015).

3.1.3.1 Rastreamento Óptico

Nessa categoria, a posição do dispositivo de captura é obtida com base em informações extraídas de sensores ópticos. Tem se tornado cada vez mais popular devido aos requisitos mínimos necessários, maior eficácia computacional dos dispositivos que a utilizam e a onipresença de um tipo desses dispositivos, os smartphones, uma plataforma ideal para RA por possuir câmera e tela (Billinghurst, M.; Clark, A.; Lee, G., 2015). Dentre as formas de realizar o rastreio com sensores ópticos temos a extração de informações baseada no espectro de luz visível e infravermelho.

O rastreamento óptico utilizando o espectro de luz visível é uma forma comumente utilizada em aplicações RA para dispositivos móveis. Dispondo da câmera do aparelho, essas aplicações identificam marcadores posicionados ou características naturais no ambiente para realização do rastreio. Para facilitar a identificação dos marcadores estes são muitas vezes compostos por somente 2 tons, como na Figura 6,

reduzindo a sensibilidade a variações nas condições luminosas do ambiente (HIRZER, 2008).

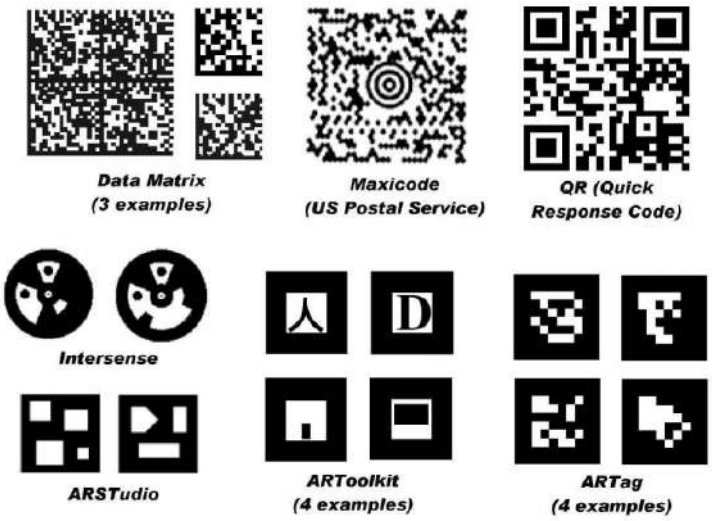

Figura 6 – Exemplos de marcadores utilizados em RA
(FIALA, 2005)

Uma biblioteca de software de RA que utiliza marcadores é o ARToolKit (Kato, H.; Billinghurst, M., 1999). Nela são identificadas bordas na imagem capturada do ambiente que podem pertencer ao marcador. Todas as regiões delimitadas por 4 linhas são consideradas potenciais marcadores, estas são utilizadas para calcular matrizes de transformação que ao serem aplicadas na imagem original do marcador torna possível compará-lo e identificá-lo (Billinghurst, M.; Clark, A.; Lee, G., 2015). A Figura 7 mostra o processo simplificado do ARToolKit.

Em alguns contextos o uso de marcadores pode ser considerado intrusivo, já

Figura 7 – Técnica de identificação de marcadores e renderização do ARToolKit
(BILLINGHURST et al., 2000)

que muitas vezes precisa que o usuário modifique o ambiente real com a inserção dos marcadores antes de usufruir da aplicação de RA. Com os avanços da área de visão computacional e dispositivos de captura, vêm se tornado cada vez mais prático o rastreamento a partir de características naturais presentes no ambiente real. Tal prática possui ainda mais desafios, como por exemplo: qual procedimento deve ser adotado para lidar com cenários de larga escala, objetos pequenos, condições variadas de iluminação, materiais com baixa texturização, transparentes ou reflexivos (LIMA et al., 2017). Os algoritmos desse tipo de rastreamento determinam características na imagem que são únicas a seus arredores, como pontos, bordas e a interseção de linhas, e delas realizam cálculos capazes de identificar o objeto a ser rastreado (Billinghurst, M.; Clark, A.; Lee, G., 2015). Alguns desses algortimos são o SIFT (LOWE, 2001), SURF (BAY; TUYTELAARS; GOOL, 2006) e BRIEF (CALONDER et al., 2010).

No rastreamento óptico por infravermelho o rastreio pode ser feito de duas maneiras: emissores de luz infravermelha são colocados no objeto a ser rastreado e câmeras são posicionadas no ambiente ao seu redor, prática conhecida como "outside-looking-in" (RIBO; PINZ; FUHRMANN, 2001), ou luzes infravermelhas são posicionadas no ambiente enquanto câmeras são acopladas no objeto a ser rastreado, prática conhecida como "inside-looking-out" (GOTTSCHALK; HUGHES, 1993) (Billinghurst, M.; Clark, A.; Lee, G., 2015). Em RA, esse objeto a ser rastreado pode ser o HMD utilizado pelo usuário. Embora sistemas de rastreamento desse tipo sejam escaláveis, podendo rastrear vários objetos ao mesmo tempo com precisão, eles também são complexos e intrusivos, devido a sua infraestrutura, além de financeiramente custosos (Billinghurst, M.; Clark, A.; Lee, G., 2015) (PINTARIC; KAUFMANN, 2007).

Utilizando-se de um sensor de profundidade é possível extrair informações tridimensionais de um objeto ou ambiente. Um exemplo é o sistema KinectFusion (IZADI et al., 2011), mostrado na Figura 8, que faz uso do mapa de profundidade produzido por um dispositivo Kinect (MICROSOFT, 2010) em movimento para gerar uma estrutura 3D detalhada de um ambiente. O sensor de profundidade do Kinect é composto por um projetor infravermelho junto com uma câmera de infravermelho, utilizando do princípio de luz estruturada para produzir o mapa de profundidade (ZHANG, 2012). Usufruindo do sensor de profundidade em conjunto com a câmera RGB do Kinect, para captura das cores do ambiente, é possível produzir um cenário de RA como em (IZADI et al., 2011) e (LIMA et al., 2017).

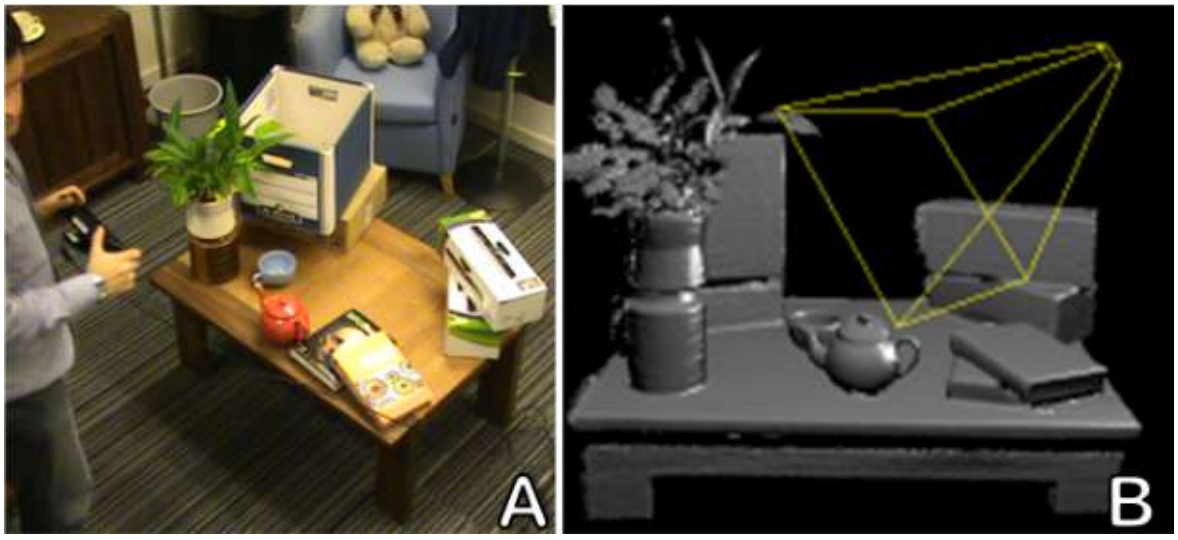

Figura 8 – KinectFusion possibilita o registro e rastreio com estruturas 3D complexas. A) Usuário movimenta o Kinect ao redor da mesa. B) Modelo 3D gerado a partir dos dados coletados através Kinect (a estrutura amarela representa o atual campo de visão sendo registrado pelo Kinect)
(IZADI et al., 2011)

3.2 Fotorrealismo

Na arte, o fotorrealismo é um estilo reconhecido pela aproximação das características de uma obra com os de uma fotografia do mundo real. Não se deve confundir com o realismo, estilo originado na segunda metade do século XIX conhecido por relevar as emoções e representar a realidade de forma mais objetiva (ENCICLOPÉDIA ITAú CULTURAL DE ARTE E CULTURA BRASILEIRAS, 2019). Na Figura 9 temos uma das pinturas fotorrealistas de Charles Bell, onde a riqueza dos detalhes aproxima muito a pintura à uma fotografia de um ambiente real.

Embora inicialmente um estilo artístico de pinturas, o termo fotorrealista vêm sendo utilizado para descrever cenários criados em motores gráficos e jogos digitais (EPIC GAMES, 2019) (STUART, 2015). Melhor capacidade gráfica não implica diretamente em gráficos ao estilo fotorrealista, mas sim mais detalhado, que como consequência possibilita que tal estilo seja replicado com maior qualidade. Na Figura 10 temos dois jogos que utilizam estilos gráficos 3D bem distintos do fotorrealismo, mas que também usufruem de capacidades gráficas avançadas. O uso de estilos gráficos diferentes é algo também existente em RA. Como podemos verificar em (REGENBRECHT et al., 2017) e (Fischer; Bartz; Straber, 2005), modificar a forma que o ambiente real é renderizado em conjunto com os elementos virtuais é uma alternativa para amplificar a imersão do usuário, se distanciando do fotorrealismo.

A complexidade de um modelo ou ambiente 3D a ser renderizado depende de uma gama de fatores, entre eles: a sua quantidade de polígonos, as texturas aplicadas, a luz e iluminação. Modelos e texturas fotorrealistas podem ser extraídos de escaneamentos de um ambiente real, como em (EPIC GAMES, 2019). A iluminação do ambi-

Figura 9 – Pintura em óleo Gumball II, de Charles Bell. A representação de efeitos como sombras, distorções e reflexos da luz no vidro se assemelham muito aos encontrados em ambientes reais.
(WIKIART VISUAL ART ENCYCLOPEDIA, 2012)

Figura 10 – The Legend of Zelda: Wind Waker (esquerda) utiliza de um estilo gráfico cartunesco, criando imagens que parecem pertencer a um desenho animado. Okami (direita) possui um estilo gráfico inspirado em um estilo de pintura japonês.
(THE TELEGRAPH, 2011)

ente depende do tipo de luz a ser utilizada e dos efeitos que os objetos renderizados devem aplicar sobre ela.

3.2.1 Efeitos derivados de atuadores invisíveis ou do meio de captura

Buscando replicar as propriedades de uma captura de um ambiente real, além dos efeitos sofridos pela luz por conta dos objetos visíveis presentes, um cenário fotorrealista pode incluir efeitos consequentes de atuadores invisíveis. Em um ambiente real esses atuadores invisíveis afetam a luz em seu trajeto para a câmera. O efeito de miragem causado quando a luz viaja por camadas de ar quente, por exemplo, pode ser renderizado em cenários com possíveis intensas fontes de calor, como na Figura 11.

Câmeras digitais capturam imagens através da exposição do Dispositivo de Carga Acoplada, também conhecido como CCD (*Charge Coupled Device*), mostrado na Figura 12. A exposição do CCD é controlada pelo obturador, que quando aberto permite a entrada de luz e enquanto fechado a impede. Dependendo da sensibilidade e do tempo de exposição do CCD alguns efeitos podem ser registrados na imagem, entre eles o ruído e os borrões de movimento. O ruído é mais facilmente percebido em imagens escuras, como na Figura 13, registrando alguns pixels com uma coloração imprecisa, dando uma impressão granulada. Os borrões de movimento são registrados quando um ou mais elementos da imagem estão se movendo em relação a câmera, isto é, se o próprio elemento está se movendo em uma direção enquanto a câmera se move em outra (ou permanece parada) e vice-versa. Pode ser percebido em forma de mancha distorcendo a imagem, como na Figura 14. Esse efeito ocorre quando a movimentação se dá durante o breve período em que o CCD é exposto, registrando a luz refletida pelos elementos que se moveram em relação a câmera pelo seu trajeto.

Figura 11 – Cena do jogo F.E.A.R. onde é replicado o efeito de miragem ao redor da explosão, que em um ambiente real seria causado pelo calor emitido. (BERILLO, 2011)

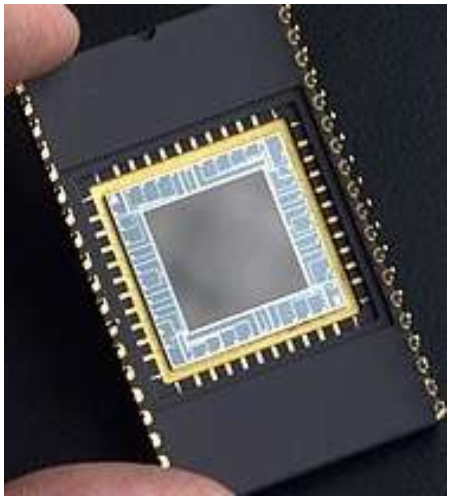

Figura 12 – Dispositivo de carga acoplada (ENFITEC JúNIOR, 2018)

3.3 Funções Matemáticas

No desenvolvimento desse trabalho o conhecimento de duas funções matemáticas foram necessárias.

3.3.1 Interpolação linear entre dois pontos

Dada as coordenadas de dois pontos num sistema 2D, pode-se obter as coordenadas de quaisquer pontos entre esses dois pontos através da interpolação linear. Na Figura 15 podemos observar uma linha traçada do ponto A ao ponto B. Essa linha é chamada de interpolante, e as coordenadas (X, Y) dos pontos C, D e E podem ser

Figura 13 – Fotografia com quantidade considerável de ruído, dando uma impressão granulada a imagem.
(CAPTURE THE ATLAS, 2019)

Figura 14 – Fotografia com borrões de movimento registrado da locomoção do ônibus.
(PRADA, 2008)

calculadas da seguinte forma:

$$(X, Y) \begin{cases} X = X_a + i(X_b - X_a), \\ Y = Y_a + i(Y_b - Y_a), \end{cases}$$

Onde i corresponde a um valor entre 0 e 1 dependente da proximidade do ponto a ser encontrado em relação aos pontos de coordenadas conhecidas (X_a, Y_a) e (X_b, Y_b). Quanto mais próximo o valor de i estiver de 0, mais próximo do ponto A vai estar, assim como quanto mais se aproximar de 1 para o ponto B. Considerando o ponto C da Figura 15 como exemplo, podemos encontrar suas coordenadas com i =

0.5, visto que o ponto está localizado exatamente no meio da interpolante.

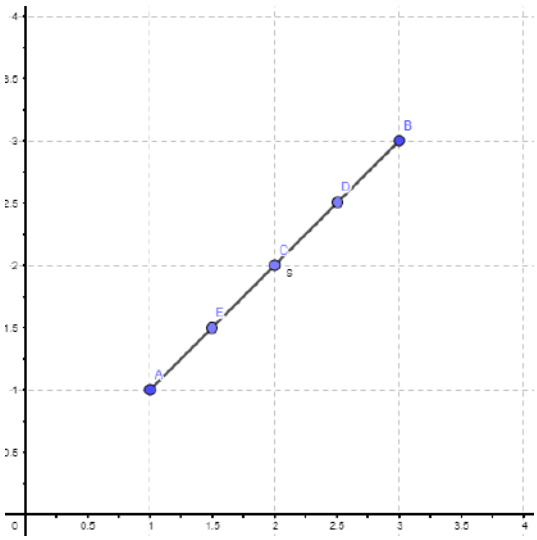

Figura 15 – Sistema de coordenadas 2D. As coordenadas dos pontos C, D e E podem ser obtidas por interpolação linear entre os pontos A e B. Fonte: O autor

3.3.2 Transformação afim

Uma transformação afim pode ser utilizada em uma imagem para produzir outra a partir desta, conservando seus pontos, linhas, planos e paralelismo (MATHWORKS, 2019). A transformação é representada por uma matriz 3x3, que ao ser multiplicada pela matriz constituída das coordenadas de um ponto na imagem original, obtém como resultado uma matriz representando as novas coordenadas deste ponto na imagem transformada. Uma imagem pode sofrer diferentes tipos de transformações, como translação, cisalhamento, rotação, ou uma composição de uma ou mais transformações, a partir de diferentes matrizes de transformação afim. A seguir temos um exemplo para uma translação:

$$\begin{bmatrix} X_T \\ Y_T \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_a & T_b & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Aqui, X e Y representam as coordenadas de um ponto na imagem original, enquanto X_T e Y_T representam as coordenadas deste ponto na nova imagem, e T_a e T_b os valores a serem aplicados na translação.

4 Solução de Renderização Fotorrealista de Borrões de Movimento em Realidade Aumentada

O capítulo presente descreve a proposta deste trabalho. A seguinte solução foi desenvolvida no motor gráfico Unity (UNITY TECHNOLOGIES, 2019b) utilizando o Vuforia (PTC, 2019), Kit de Desenvolvimento de Software para RA integrado ao Unity. Os algoritmos desenvolvidos foram implementados nas linguagens C# e CG/HLSL baseado nos métodos e algoritmos em GLSL descritos em (Park, Y.; Lepetit, V.; Woo, W., 2012). Foi optado utilizar o Unity devido a sua integração nativa com o Vuforia, disponibilização gratuita e sua popularidade, o que facilita o aprendizado quanto ao uso da ferramenta dada a variedade de conteúdo encontrado sobre a mesma na Internet.

Nas próximas seções são descritas as ferramentas utilizadas e em seguida o método para renderização de borrões de movimentos fotorrealistas em RA no Unity.

4.1 Unity

Motor gráfico multi-plataforma desenvolvido pela Unity Technologies, o Unity pode ser utilizado no desenvolvimento de aplicações 3D, 2D, em RV e RM. Sendo um dos motores gráficos mais populares na criação de jogos, também é utilizado em outras áreas como animação, automotiva, arquitetura e engenharia. Conta com uma Interface de Programação de Aplicativos em C# no qual os usuários implementam as rotinas necessárias para suas aplicações. O Unity também possibilita que softwares desenvolvidos neste possam ser implantados em mais de 25 plataformas diferentes, desde dispositivos móveis, a consoles de videogames e computadores, dadas as limitações de hardware.

Como podemos ver na Figura 16 a interface do Unity permite ao usuário visualizar o ambiente renderizado enquanto o edita. Ela possui um editor em hierarquia dos objetos no cenário, presente na parte esquerda, e um inspetor, na parte direita, que possibilita a visualização e edição das propriedades e rotinas atreladas ao objeto selecionado. O usuário também pode executar sua aplicação pela aba “Game” ao mesmo tempo que a modifica, praticando testes em tempo real sem a necessidade de reiniciar o sistema.

Figura 16 – Interface do motor gráfico Unity
(UNITY TECHNOLOGIES, 2019b)

4.2 Vuforia

Pacote de desenvolvimento de RA e RM utilizado para desenvolver aplicações para diversos setores como industrial, entretenimento, ciências naturais e automotivo. Compatível com dispositivos móveis e HMDs, o Vuforia utiliza tecnologia de visão computacional para reconhecimento e rastreio de uma variedade de alvos customizáveis. A partir das últimas versões do Unity, o Vuforia passou a vir integrado neste.

No portal do desenvolvedor Vuforia, na Internet, é possível registrar novos alvos, podendo este ter forma de plano, cuboide, cilindro ou outro objeto 3D. Ao adicionar uma imagem a ser utilizada como alvo, esta é processada e avaliada com uma nota de 0 a 5 dependendo do quão fácil é de rastreá-la. A facilidade de rastreio depende das características da imagem identificadas pelo Vuforia, principalmente se estas possuem detalhes únicos, como na Figura 17.

4.3 Shaders no Unity

Shaders são rotinas executadas pela Unidade de Processamento Gráfico responsáveis por desenhar os pixels na tela. Para replicar efeitos como o borrão de movimento é necessário programar um *shader*.

Na Figura 18 temos uma representação simplificada do processo de renderização de cenários do Unity. Um cenário é composto por modelos 3D. Cada modelo 3D é definido por vértices posicionados no ambiente, as cores desses vértices e seus vetores normais, que estabelecem o lado interior e exterior do modelo. Esses modelos

Figura 17 – Imagem a ser utilizada no alvo pelo Vuforia (a esquerda) e a mesma imagem após ser processada (a direita), exibindo suas características (em amarelo) a serem utilizadas no rastreamento.
(PTC, 2019)

também possuem dados de mapeamento UV, responsáveis por determinar como uma imagem de textura 2D deve ser aplicada ao modelo. Cada modelo possui um material composto por uma textura 2D e valores de propriedades a serem utilizados no processo de renderização do *shader*, como parâmetros para intensidade de efeitos de reflexão da luz ou transparência. Por fim, um material utiliza um *shader*, que é composto por uma ou mais rotinas nas linguagens de programação CG e HLSL, determinando como cada pixel na tela que represente parte do modelo 3D visualizado deve ser pintado.

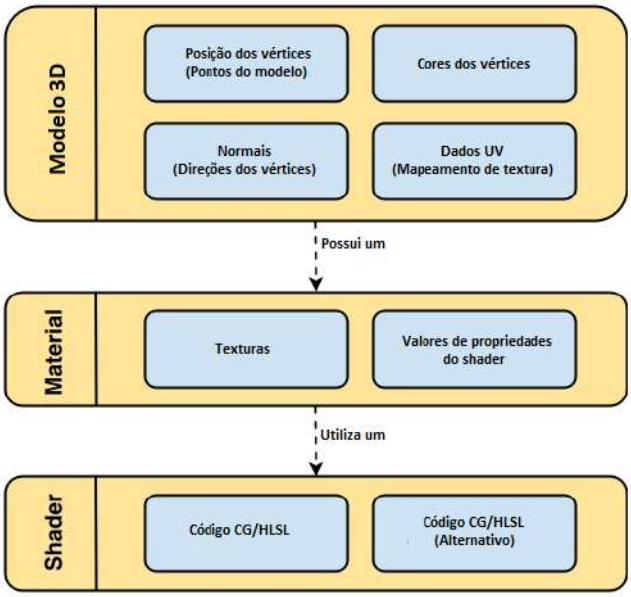

Figura 18 – Processo de renderização simplificado do Unity.
(UNITY TECHNOLOGIES, 2019a)

4.4 Renderização de Borrões de Movimento

É possível replicar o efeito de borrão de movimento em um objeto virtual ao aplicar um *shader* de borrão simples e sincronizando a intensidade do borrão nos eixos

horizontal e vertical com a velocidade do elemento virtual em relação à câmera. Tal efeito porém não geraria borrões adequados para movimentos de rotação do objeto. Para reproduzir borrões de movimento de forma realista é preciso que o objeto virtual deixe um "rastro" borrado que demonstre seu movimento em relação ao ponto de vista. Uma possível maneira de alcançar isto seria através de múltiplas renderizações do elemento virtual pelo trajeto da movimentação, seguida pela mistura das cores dos pixels dessas renderizações para criação do borrão. Com muitas renderizações do elemento 3D em sequência essa aproximação acaba sendo custosa especialmente para RA, onde ela deve ser feita em tempo real, podendo ocorrer gargalos por falta de recursos do sistema que prejudicam a experiência do usuário.

Buscando por um método mais eficaz de renderizar o borrão de movimento, foi optado neste trabalho adotar uma estratégia similar a utilizada em (Park, Y.; Lepetit, V.; Woo, W., 2012). A técnica consiste na renderização 3D do objeto virtual em 2 posições diferentes ao longo do movimento realizado pelo objeto virtual durante um período de tempo que representa a duração em que o obturador da câmera está aberto. Para cada uma dessas posições é produzido um quadro denominado *Intraframe*. Para complementar os *Intraframes* é gerada uma quantidade de imagens intermediárias a partir de transformações afins aplicadas a esses *Intraframes*. Essas imagens intermediárias são chamadas aqui de *Interframes*. Na Figura 19 temos uma representação de como essas imagens são utilizadas em (Park, Y.; Lepetit, V.; Woo, W., 2012). São geradas duas imagens, (c) e (e), ao renderizar o objeto 3D durante o trajeto de seu movimento chamados de *Intraframes*. Para criar as outras imagens necessárias são aplicadas transformações afins nos *Intraframes*. (b) foi gerada da transformação de (c) enquanto (f) da transformação de (e). Já para as imagens intermediárias entre (c) e (e), é feita a transformação de cada *Intraframe*, aplicado um peso a essas transformações e então somando-as, obtendo uma imagem como (d). Ao somar todas as imagens obtidas é produzido um borrão como (a). Relacionando essas imagens da Figura 19 com a Figura 20, pode-se dizer que (c) e (e) representam respectivamente as posições demarcadas como $\frac{1}{4}$ e $\frac{3}{4}$ do trajeto, (b) representa uma parte qualquer antes de $\frac{1}{4}$ do trajeto assim como (f) representa uma parte depois de $\frac{3}{4}$ do trajeto, enquanto (d) representa uma parte entre as duas marcações.

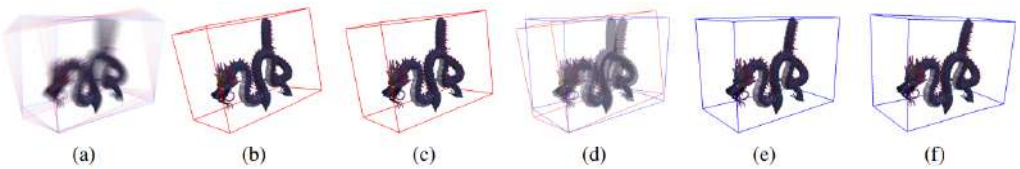

Figura 19 – Para reproduzir o borrão de movimento em (a) são geradas e somadas imagens intermediárias como de (b) a (f).
(Park, Y.; Lepetit, V.; Woo, W., 2012)

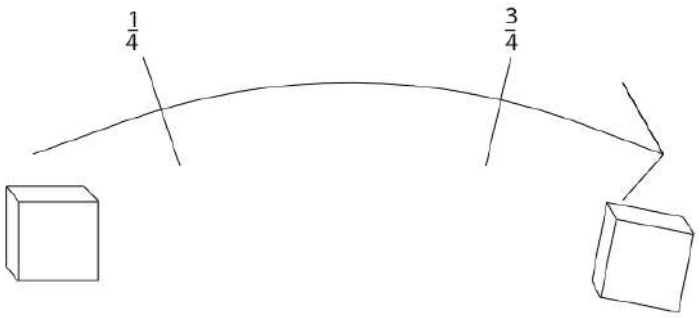

Figura 20 – Representação de um trajeto realizado por um objeto durante a exposição do dispositivo de carga acoplada, quando deve ser registrado um borrão de movimento. Fonte: O autor

4.4.1 Produzindo as imagens intermediárias: *Intraframes* e *Interframes*

Como já mencionado, para a geração dos 2 *Intraframes* são consideradas duas posições ao longo do trajeto percorrido pelo elemento virtual durante um tempo que representa a duração em que o obturador da câmera está aberto. Essas duas posições representam $\frac{1}{4}$ e $\frac{3}{4}$ da diferença do posicionamento do elemento antes e depois do movimento realizado, e são renderizadas em tempo real. Para a geração dos *Interframes* é preciso antes determinar a quantidade total de imagens intermediárias necessárias (*Intraframes* e *Interframes*).

Definimos a quantidade de imagens intermediárias criando uma caixa retangular cujo as extremidades são delimitadas pelo elemento virtual a ser renderizado, de maneira que o inclua inteiramente no menor volume possível. É calculada a distância em pixels da tela que os 4 vértices da base dessa caixa percorrem durante o período estimado em que o obturador está aberto. É então retirada a média da soma dessas distâncias. O resultado desses cálculos determina a quantidade de *Interframes*. Para fins de eficiência foi estabelecido o limite máximo de 48 *Interframes*, que somados com os 2 *Intraframes* totalizam ao máximo 50 imagens intermediárias, sendo o suficiente para renderização de um borrão fotorrealista na ocorrência de borrões intensos do ambiente onde o alvo pode ser rastreado pelo Vuforia.

Com a quantidade de imagens intermediárias determinada é calculado o piso de sua divisão por 4. O valor resultante define a quantidade de imagens intermediárias para a geração do borrão em 3 intervalos delimitados pelos *Intraframes*. Chamando a quantidade de imagens intermediárias de n, podemos considerar o resultado obtido como $\frac{n}{4}$. Chamando também a diferença entre o posicionamento do elemento virtual no tempo em que o obturador da câmera está aberto de d, as imagens intermediárias durante esse período ficam então definidas como:

- $\frac{n}{4}$ imagens intermediárias para o posicionamento do elemento virtual do mo-

mento em que o obturador da câmera começa a ser aberto até sua posição em $\frac{d}{4}$. Sendo as primeiras $\frac{n}{4} - 1$ imagens *Interframes* e a última o *Intraframe* correspondente a $\frac{d}{4}$;

- $\frac{2n}{4}$ imagens intermediárias compostas por *Interframes* correspondentes ao posicionamento do objeto entre $\frac{d}{4}$ e $\frac{3d}{4}$;
- $\frac{n}{4}$ imagens intermediárias, com a primeira sendo o *Intraframe* correspondente a $\frac{3d}{4}$ e o restante *Interframes* correspondentes ao posicionamento entre $\frac{3d}{4}$ e d.

Considerando esses 3 intervalos calculamos o *Interframe* F_i para i variando de 1 a n, excluindo os valores $\frac{n}{4}$ e $\frac{3n}{4}$ onde são utilizados os *Intraframes*, da seguinte forma:

$$F_i = \begin{cases} T(I_{\frac{n}{4}}, A_{\frac{n}{4} \to i}), & \text{se } i < \frac{n}{4} \\ \alpha(i)T(I_{\frac{n}{4}}, A_{\frac{n}{4} \to i}) + \beta(i)T(I_{\frac{3n}{4}}, A_{\frac{3n}{4} \to i}), & \text{se } \frac{n}{4} < i < \frac{3n}{4} \\ T(I_{\frac{3n}{4}}, A_{\frac{3n}{4} \to i}), & \text{se } i > \frac{3n}{4} \end{cases}$$

Onde $A_{j \to k}$ representa uma matriz de transformação afim que quando aplicada a imagem intermediária I_j (*Intraframe*) gera o *Interframe* correspondente a F_k. $T(\ ,\ )$ representa a aplicação da matriz de transformação. $\alpha(i)$ e $\beta(i)$ variam linearmente respectivamente de 1 a 0 e de 0 a 1 conforme i varia de $\frac{n}{4}$ para $\frac{3n}{4}$.

Para encontrar a matriz de transformação afim $A_{j \to k}$ utilizamos a posição dos vértices do retângulo da base da caixa utilizada anteriormente para calcular a quantidade de imagens intermediárias. Realizando operações de interpolação linear entre as coordenadas dos vértices do retângulo no instante 0 em que o obturador da câmera começa a abrir e no instante 1 em que a imagem foi capturada, considerando as n imagens a serem registradas, conseguimos estimar as coordenadas dos vértices do retângulo em qualquer uma dessas imagens. Com essas coordenadas convertidas em coordenadas de tela podemos calcular a matriz de transformação afim que ao ser aplicada nas coordenadas de tela do retângulo base no *Intraframe* I_j a transformam nas primeiras, respectivas ao *Interframe* F_k. Aplicando as matrizes de transformações afins obtidas em cada pixel dos *Intraframes* definimos $T(I_j\ ,\ A_{j \to k})$.

Após a obtenção dos *Intraframes* e *Interframes* todos são somados a uma imagem vazia. Todo pixel cujo valor continuar vazio não foi preenchido por nenhuma imagem intermediária e deve ser pintado com sua cor respectiva à imagem capturada pela câmera. Os pixels cuja quantidade de imagens intermediárias somadas nestes for igual

à quantidade total de imagens intermediárias pertencem totalmente ao objeto e são pintados com a média da soma das cores desse pixel nas imagens intermediárias. Já os pixels cujo a quantidade de imagens intermediárias somadas a estes é menor que o total de imagens intermediárias pertencem ao “rastro” do borrão de movimento do objeto, e são pintados com uma cor calculada a partir da interpolação linear entre a média da cor do pixel da soma das imagens intermediárias e a cor do pixel vinda da imagem capturada pela câmera. Este último se aproxima mais da cor do pixel vindo da câmera quanto menor a quantidade de imagens intermediárias somadas a este, assim como se aproxima da cor média do pixel respectivo da soma das imagens intermediárias para o quanto maior for a quantidade dessas imagens somadas em suas coordenadas. Na Figura 21 temos um exemplo mostrando os pixels cuja as cores são calculadas a partir de maneiras diferentes.

Figura 21 – A esquerda temos uma renderização onde, em azul, estão todos os pixels cujo a cor deve ser calculada a partir da média da soma das cores pertencentes somente ao objeto virtual vindo das imagens intermediárias, enquanto em verde estão aqueles que devem utilizar a imagem capturada do ambiente real em seu cálculo. Na direita temos uma renderização do objeto virtual com borrão de movimento realizando esses cálculos. Fonte: O autor

4.4.2 Implementação

Para a execução de uma aplicação simples de RA no Unity é necessário criar uma cena com uma câmera RA Vuforia, um alvo Vuforia e um ou mais objetos quaisquer como filhos do alvo no sistema de hierarquia. Nesse sistema, qualquer objeto alocado como filho de um segundo objeto vai seguir quaisquer mudanças de posicionamento deste, como se o primeiro objeto estivesse ”ancorado”ao segundo. Nessa cena, os objetos virtuais alocados como filhos do alvo Vuforia seguirão a sua movimentação durante o rastreio.

Como explicado na subseção anterior, o primeiro passo para a obtenção de todas as imagens intermediárias a serem somadas é a renderização dos *Intraframes*. Para cumprir esse objetivo são utilizadas duas câmeras adicionais posicionadas no mesmo local que a câmera RA na cena. Cada uma delas é configurada para observar objetos pertencentes a uma camada específica diferente. É feito o uso de uma rotina principal, na câmera RA, que se comunicará com os *shaders* e uma rotina auxiliar, no alvo, que executa a rotina principal enquanto o alvo está sendo rastreado pela câmera. Essa rotina principal é executada em um laço com intervalo de tempo parametrizado entre cada iteração. O intervalo corresponde ao tempo em segundos que a câmera toma para abrir e fechar o obturador. Ainda na rotina principal são instanciadas duas cópias do objeto a ser renderizado em RA, cada uma pertencendo a camadas diferentes definidas para as câmeras auxiliares. Entre cada intervalo essas cópias são reposicionadas através da interpolação linear utilizando a rotação e posição do objeto antes e depois para encontrar os posicionamentos respectivos a $\frac{1}{4}$ e $\frac{3}{4}$ da diferença da localização do objeto no período. Com cada uma das câmeras auxiliares observando somente uma cópia do objeto temos então os dois *Intraframes*.

Os *Interframes* são desenhados no *shader* utilizando matrizes de transformações e os *Intraframes*. As matrizes são calculadas na rotina principal através do posicionamento dos vértices do retângulo base da caixa, como explicado na subseção anterior. O algoritmo 1 foi utilizado para encontrar as matrizes. Vemos na Figura 22 que são alocados 4 cubos, um para cada vértice. Esses cubos são configurados como filhos do alvo e estão em uma camada invisível à câmera RA. Como no *shader* a imagem a ser desenhada é tratada pixel por pixel, sempre considerando o ponto atual a ser desenhado, não é possível aplicar uma matriz a um *Intraframe* como um todo. Para contornar esse problema é feito a inversão da matriz antes de repassá-la para o *shader*. Com a matriz inversa, para cada ponto do *Intraframe* que deveria ser desenhado no pixel considerado atualmente pelo *shader* é feita uma transformação nas coordenadas do ponto considerado. Acessando a cor do novo ponto considerado e a pintando no pixel temos como resultado um pixel pertencente ao *Interframe*, respectivo à transformação do *Intraframe*.

Algoritmo 1 Encontrar matriz de transformação afim

```
 1: função matrizAfim3Pontos2D(double[] a1, b1, c1, a2, b2, c2)
 2:     double[,] matrizAfim = new double[3,3]
 3:     double[,] matrizX = new double[6,6]
 4:     matrizX[1,1] = a1[1]
 5:     matrizX[1,2] = a1[2]
 6:     matrizX[1,3] = 1
 7:     matrizX[1,4] = 0
 8:     matrizX[1,5] = 0
 9:     matrizX[1,6] = 0
10:     matrizX[2,1] = 0
11:     matrizX[2,2] = 0
12:     matrizX[2,3] = 0
13:     matrizX[2,4] = a1[1]
14:     matrizX[2,5] = a1[2]
15:     matrizX[2,6] = 1
16:     matrizX[3,1] = b1[1]
17:     matrizX[3,2] = b1[2]
18:     matrizX[3,3] = 1
19:     matrizX[3,4] = 0
20:     matrizX[3,5] = 0
21:     matrizX[3,6] = 0
22:     matrizX[4,1] = 0
23:     matrizX[4,2] = 0
24:     matrizX[4,3] = 0
25:     matrizX[4,4] = b1[1]
26:     matrizX[4,5] = b1[2]
27:     matrizX[4,6] = 1
28:     matrizX[5,1] = c1[1]
29:     matrizX[5,2] = c1[2]
30:     matrizX[5,3] = 1
31:     matrizX[5,4] = 0
32:     matrizX[5,5] = 0
33:     matrizX[5,6] = 0
34:     matrizX[6,1] = 0
35:     matrizX[6,2] = 0
36:     matrizX[6,3] = 0
37:     matrizX[6,4] = c1[1]
38:     matrizX[6,5] = c1[2]
39:     matrizX[6,6] = 1
```

Para desenhar a imagem final na tela, o *shader* recebe da rotina principal: a textura da imagem do ambiente real capturado pela câmera, as texturas dos 2 *Intraframes*, uma textura somente com a renderização do objeto virtual em sua posição atual, a quantidade de imagens intermediárias juntamente com o piso de sua divisão por 4 e as matrizes de transformações inversas. Para obter a textura com somente a ren-

```
40:     double[,] xPrime = new double[6,1]
41:     xPrime[1,1] = a2[1]
42:     xPrime[2,1] = a2[2]
43:     xPrime[3,1] = b2[1]
44:     xPrime[4,1] = b2[2]
45:     xPrime[5,1] = c2[1]
46:     xPrime[6,1] = c2[2]
47:     double[,] matrizColuna = multiplicarMatrizes(matrizInversa(matrizX), xPrime)
48:     matrizAfim[1,1] = matrizColuna[1,1]
49:     matrizAfim[1,2] = matrizColuna[2,1]
50:     matrizAfim[1,3] = matrizColuna[3,1]
51:     matrizAfim[2,1] = matrizColuna[4,1]
52:     matrizAfim[2,2] = matrizColuna[5,1]
53:     matrizAfim[2,3] = matrizColuna[6,1]
54:     matrizAfim[3,1] = 0
55:     matrizAfim[3,2] = 0
56:     matrizAfim[3,3] = 1
57:     devolve matrizAfim
58: fim função
```

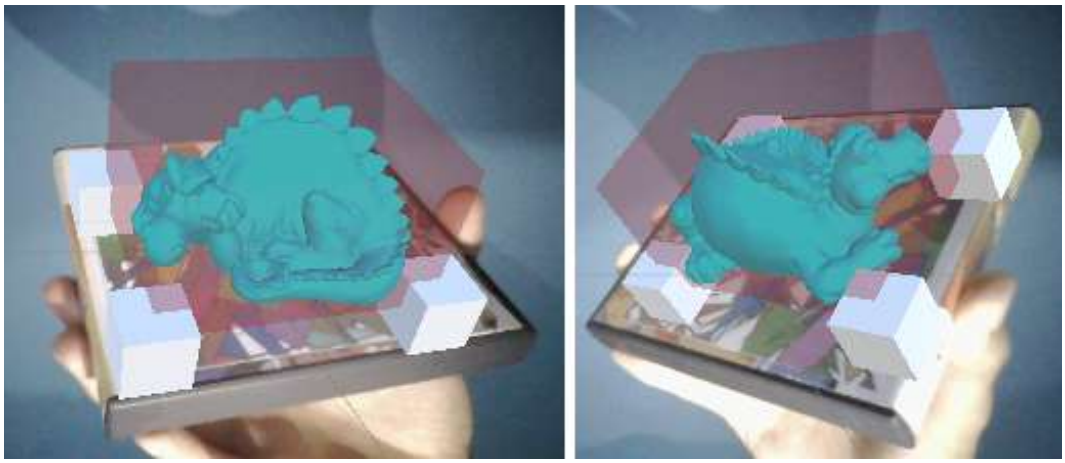

Figura 22 – Caixa retangular (em vermelho semitransparente) com seus vértices do retângulo base sobrepostos por cubos brancos. A caixa contem o objeto virtual a ser renderizado. Fonte: O autor

derização do objeto virtual em sua posição atual é preciso criar uma terceira câmera auxiliar, similar as outras duas, configurada para observar somente o objeto, enquanto para registrar somente o ambiente real capturado pela câmera é necessário que a câmera RA seja ajustada para não registrar o objeto virtual. No *shader* são utilizados 4 canais para cada pixel a ser pintado, sendo 3 canais de cores (vermelho, verde e azul) e o quarto o alfa (transparência). No *shader* de fragmentos, onde fica o código responsável por definir como cada pixel da tela deve ser pintado, é feita uma rotina em laço para somar a cor do ponto em consideração de todas as texturas das imagens intermediárias e da textura com a renderização do objeto em sua posição atual. Cada

cor a ser somada, se for originada de um ponto pertencente ao objeto virtual terá seu alfa igual a 1 (sem transparência), enquanto no caso contrário, quando o ponto não pertence ao objeto, todos os 4 canais estarão zerados. Como resultado temos que o canal alfa do pixel pintado guarda um valor igual à quantidade de imagens somadas neste. Essa propriedade é aproveitada para definir como as cores dos pixels do objeto virtual devem se misturar com as cores da imagem registrada pela câmera por interpolação linear. Ao fim do processamento por esse primeiro *shader*, há uma outra passada por um segundo *shader* responsável somente por preencher os pixels vazios resultantes do primeiro *shader* com a imagem capturada pela câmera. O algoritmo 2 foi utilizado no primeiro *shader*.

Algoritmo 2 Shader de fragmentação

```
1:  função frag(v2f i)
2:      float4 col = float4(0, 0, 0, 0)
3:      float4 bg = tex2D(CamBg, i.uv)                          ▷ Imagem da câmera
4:      float4 intraF1 = tex2D(intraFrame1, i.uv)               ▷ Intraframe respectivo a 1/4
5:      float4 intraF3 = tex2D(intraFrame3, i.uv)               ▷ Intraframe respectivo a 3/4
6:      float4 coldf = tex2D(MainTex, i.uv)                     ▷ Objeto virtual na posição atual
7:      float beta = 1.0 / (interQtd * 2.0)
8:      para i ← 1 até framesQtd faça                           ▷ Quantidade de imagens intermediárias
9:          se i < interQtd então                               ▷ interQtd é o piso de 1/4 de framesQtd
10:             float2 uvTransformado = transformarPorMatriz(i.uv, matrizInversa[i])
11:             col = col + tex2D(intraFrame1, uvTransformado)
12:         fim se
13:         se i = interQtd então
14:             col = col + intraF1
15:         fim se
16:         se interQtd * 3 > i > interQtd então
17:             float alfa = 1.0 - beta
18:             float2 uvTransformadoAlfa = transformarPorMatriz(i.uv,
19:                 matrizInversaAlfa[i])
20:             float2 uvTransformadoBeta = transformarPorMatriz(i.uv,
21:                 matrizInversaBeta[i])
22:             col = col + alfa * tex2D(intraFrame1, uvTransformadoAlfa) + beta *
23:                 tex2D(intraFrame3, uvTransformadoBeta)
24:             beta = beta + 1.0 / (interQtd * 2.0)
25:         fim se
26:         se i = interQtd * 3 então
27:             col = col + intraF3
28:         fim se
29:         se i > interQtd * 3 então
30:             float2 uvTransformado = transformarPorMatriz(i.uv, matrizInversa[i])
31:             col = col + tex2D(intraFrame3, uvTransformado)
32:         fim se
33:     fim para
34:     col = col + coldf
35:     se col.a = 0 então
36:         devolve 0                                           ▷ A ser pintado no segundo shader
37:     fim se
38:     se col.a > framesQtd então
39:         col = col / (col.a)
40:     senão
41:         float nInter = col.a
42:         col = col / nInter
43:         col = float4(lerp(bg, col, nInter / (framesQtd + 1)))
44:     fim se
45:     devolve col
46: fim função
```

4.5 Resultados experimentais

Os testes a seguir foram realizados em um computador notebook Sony Vaio SVE141D11L, com uma CPU de 2.50GHz e o adaptador de vídeo Intel HD Graphics 3000. A câmera utilizada, embutida no computador, captura imagens numa resolução de 640x480 a 30 quadros por segundo. O modelo do dragão é composto por 240.057 vértices e 480.076 faces, e pode ser obtido em (ACADEMY OF SCIENCES OF THE CZECH REPUBLIC AND CZECH TECHNICAL UNIVERSITY IN PRAGUE, 2011), o modelo do tatu em pé está disponível no repositório de escaneamentos 3D de Stanford (STANFORD COMPUTER GRAPHICS LABORATORY, 2014), é composto por 173.006 vertices e 345.944 faces, e o modelo do dragão em espiral é composto por 9.201 vértices e 15.959 faces, disponibilizado por Viacheslav Titenko na Unity Asset Store (UNITY TECHNOLOGIES, 2016).

Nas Figuras 23 e 24 observa-se respectivamente a renderização do objeto virtual sem e com borrões sintéticos na presença de movimentos na vertical, na horizontal e de rotação. Já na Figura 25 apresentam-se as mesmas situações para movimentos de afastamento da câmera pelo eixo de profundidade.

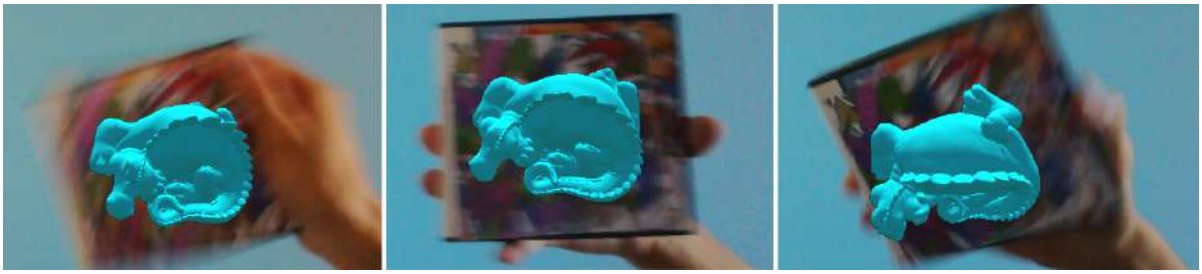

Figura 23 – Objeto virtual em RA sem a renderização de borrões de movimento. Fonte: O autor

Figura 24 – Objeto virtual em RA com a renderização de borrões de movimento deste trabalho. Fonte: O autor

O desempenho da solução no sistema utilizado é profundamente impactado pela complexidade do objeto virtual. Além do modelo 3D, as configurações da luz da cena e as propriedades do material aplicado ao objeto também afetam em sua complexidade. Todos esses fatores são considerados 3 vezes ao renderizar a cena no

Figura 25 – Objeto virtual em RA sem (esquerda) e com (direita) a renderização de borrões de movimentação com movimentos de afastamento da câmera. Fonte: O autor

Unity, uma para a projeção do objeto em sua posição atual e as outras duas para os *Intraframes*. Nas Figuras 26 e 27 encontram-se as propriedades utilizadas nos testes, configuradas para a luz da cena e o modelo do objeto, incluindo seu material.

Figura 26 – Configurações no Unity da luz da cena. Fonte: O autor

A solução foi avaliada no ambiente de teste do Unity, que disponibiliza dados do desempenho em tempo real. Executando o ambiente de RA e utilizando o modelo do dragão mostrado na Figura 24, sem a geração de borrões de movimentação sintéticos, obteve-se uma média aproximada de 60 quadros por segundo sem a renderização do objeto virtual e 50 quadros por segundo com a renderização. Já ao executar o ambiente com a geração de borrões, obteve-se um desempenho similar sem a renderização do objeto enquanto com a renderização do objeto houve uma redução para aproximadamente 25 quadros por segundo. Essa redução no desempenho não é tão perceptível visualmente, dado que a câmera registra as imagens em até 30 quadros por segundo. Testando com os modelos do tatu em pé e do dragão em espiral, mostrados respectivamente nas Figuras 28 e 29, e utilizando as mesmas propriedades de iluminação e material, foi demonstrada uma melhoria na performance, consequência da menor complexidade desses modelos. Na renderização do modelo do tatu em pé, a performance sem o uso da solução manteve-se em uma taxa de aproximadamente 50 quadros por segundo, enquanto com o uso da solução foi obtida uma taxa aproximada de 35 quadros por segundo. Já nos testes com o modelo do dragão em espiral,

Figura 27 – Configurações no Unity do modelo do objeto e o material aplicado a este. Fonte: O autor

a performance com e sem o uso da solução desenvolvida manteve-se numa taxa de aproximadamente 60 quadros por segundo.

Durante os testes houveram casos em que o borrão ou a posição do objeto não condiziam com o movimento realizado durante breves momentos. Isso pode ocorrer devido ao rastreamento do Vuforia que, embora bem otimizado, falha na presença de borrões intensos e de outros artefatos visuais que atrapalhem o registro do alvo. Pausando a aplicação pode-se visualizar a ocorrência dessas falhas, como mostrado nas Figuras 30 e 31.

Figura 28 – Modelo do tatu em pé renderizado com a solução de borrões de movimento desenvolvida. Fonte: O autor

Figura 29 – Modelo do dragão em espiral renderizado com a solução de borrões de movimento desenvolvida. Fonte: O autor

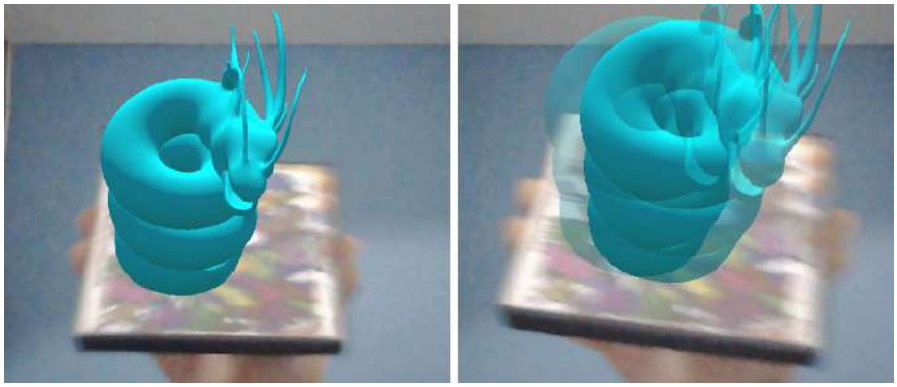

Figura 30 – Falha na reprodução do efeito de borrão de movimento. Fonte: O autor

Figura 31 – Falha no rastreamento do Vuforia causando a renderização do objeto virtual em uma posição que não corresponde a forma como este está ancorado ao alvo. Fonte: O autor

5 Conclusão

Este trabalho apresenta uma solução para a renderização de borrões de movimento fotorrealistas para RA com o uso de alvo de imagem, além de aprofundar em detalhes na implementação desta solução no motor gráfico Unity. Como visto no Capítulo 2, fotorrealismo em RA é uma linha de pesquisa com uma quantidade relevante de estudos que continua a mostrar novos desenvolvimentos em busca de resultados que cada vez mais intensificam a imersão do usuário. Tanto com pesquisas focadas nos efeitos sofridos pela luz em consequência das interações entre elementos reais e virtuais, e outras focadas nos efeitos derivados do meio de captura, nenhum dos estudos sobre RA fotorrealista encontrados havia proposto uma solução para o Unity, um dos motores gráficos mais populares no desenvolvimento de jogos.

No Capítulo 4 vimos como a solução proposta, baseada no trabalho em (Park, Y.; Lepetit, V.; Woo, W., 2012), pode ser implementada no Unity. A tecnologia de rastreio de imagens planas do kit de desenvolvimento de software Vuforia se provou eficaz para os testes com borrões moderados na imagem capturada. Outros tipos de alvos oferecidos pelo Vuforia podem ser utilizados nessa mesma solução visto que a forma que o rastreio é registrado no Unity se mantém. Seguindo as ideias desenvolvidas deve ser possível implementar essa solução em outros motores gráficos ou plataformas com suporte a RA.

A solução desenvolvida pôde ser avaliada visualmente através do ambiente de desenvolvimento e testes do Unity. Embora testado em um computador sem placa de vídeo dedicada, e renderizando um modelo 3D complexo, houve somente uma queda de aproximadamente 5 quadros por segundo em relação à taxa de captura de 30 quadros por segundo da câmera utilizada. Os borrões sintéticos produzidos se mostraram coerentes com os borrões capturados do ambiente respectivo ao movimento do marcador, com defeitos visíveis apenas na ocorrência de falhas no rastreamento.

5.1 Trabalhos Futuros

Durante os testes foi observado como a interrupção da renderização do elemento virtual por conta de borrões de movimento intensos podem prejudicar a experiência do usuário. As tecnologias de rastreio integradas ao Unity pertencem ao Vuforia e não podem ser modificadas. Sendo assim, um ponto que pode ser desenvolvido futuramente para este trabalho seria a implementação deste utilizando outros kits de desenvolvimento de software modificáveis ou com técnicas de rastreio mais precisas na ocorrência de borrões.

Outro aspecto que pode ser desenvolvido no futuro é a integração com outros métodos de renderização fotorrealistas. Considerando que o cálculo de algumas propriedades em um ambiente de RA, como reflexos, sombras e sangramento podem ser tarefas intensivas, a renderização de borrões pela solução aqui proposta triplicaria essa intensidade durante a geração dos *Intraframes*. Uma sugestão para tornar essa tarefa mais prática para a demanda da RA seria reduzir a qualidade ou desabilitar alguns efeitos que podem não ser perceptíveis na ocorrência do borrão.

Para finalizar, pretende-se disponibilizar o trabalho aqui desenvolvido em alguma plataforma aberta na internet para que possa ser continuado por terceiros ou sirva de inspiração a outros trabalhos. Também está planejado preparar a implementação no Unity para ser enviada a Unity Asset Store, tornando possível que demais usuários do motor gráfico possam importar facilmente a solução para seus projetos de RA.

Referências

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC AND CZECH TECHNICAL UNIVERSITY IN PRAGUE. *EG 2007 Phlegmatic Dragon*. 2011. Disponível em: <https://dcgi.fel.cvut.cz/cgg/eg07/index.php?page=dragon>. Acesso em: 11 nov. 2019. Citado na página 41.

ACER. *Headset Windows Mixed Reality*. 2019. Disponível em: <https://www.acer.com/ac/pt/BR/content/series/wmr>. Acesso em: 16 out. 2019. Citado na página 18.

ANIWAA. *Mixed reality vs augmented reality: what's the difference?* 2018. Disponível em: <https://www.aniwaa.com/blog/mixed-reality-vs-augmented-reality-whats-the-difference/>. Acesso em: 20 set. 2019. Citado na página 18.

APP Store. 2019. Disponível em: <https://www.apple.com/ios/app-store/>. Acesso em: 11 out. 2019. Citado na página 17.

Azuma, R. et al. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, v. 21, p. 34–47, 2001. Citado na página 18.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. In: . [S.l.: s.n.], 2006. v. 110, p. 404–417. Citado na página 22.

BERILLO, A. *Graphics Technologies in Games: F.E.A.R.* 2011. Disponível em: <http://ixbtlabs.com/articles2/video/tech_fear.html>. Acesso em: 13 nov. 2019. Citado na página 26.

Billinghurst, M.; Clark, A.; Lee, G. A survey of augmented reality. *Foundations and Trends® in Human-Computer Interaction*, v. 8, p. 73–272, 01 2015. Citado 3 vezes nas páginas 20, 21 e 22.

BILLINGHURST, M. et al. Mixing realities in shared space: An augmented reality interface for collaborative computing. In: . [S.l.: s.n.], 2000. v. 3, p. 1641–1644. Citado na página 21.

CALONDER, M. et al. Brief: Binary robust independent elementary features. In: . [S.l.: s.n.], 2010. v. 6314, p. 778–792. Citado na página 22.

CAPTURE THE ATLAS. *Noise in photography: What It is and how to correct it*. 2019. Disponível em: <https://capturetheatlas.com/noise-in-photography/>. Acesso em: 14 nov. 2019. Citado na página 27.

DOVE, J. *Apple unveils ARKit 3 for more immersive augmented reality experiences*. 2019. Disponível em: <https://www.digitaltrends.com/news/apple-unveils-arkit3-wwdc-2019/>. Acesso em: 12 dez. 2019. Citado na página 16.

EDITORA MELHORAMENTOS LTDA. *Dicionário Brasileiro da Língua Portuguesa*. 2015. Disponível em: <http://michaelis.uol.com.br/busca?r=0&f=0&t=0&palavra=real>. Acesso em: 16 out. 2019. Citado na página 17.

ENCICLOPÉDIA ITAú CULTURAL DE ARTE E CULTURA BRASILEIRAS. *Realismo*. 2019. Disponível em: <http://enciclopedia.itaucultural.org.br/termo3639/realismo>. Acesso em: 06 nov. 2019. Citado na página 24.

ENFITEC JúNIOR. *O Fotodiodo e o Armazenamento de Imagens em uma Câmera Digital*. 2018. Disponível em: <https://www.ufrgs.br/enfitecjunior/2018/05/10/o-fotodiodo-e-o-armazenamento-de-imagens-em-uma-camera-digital/>. Acesso em: 10 dez. 2019. Citado na página 26.

EPIC GAMES. *Quixel demonstrates stunning photorealism using Unreal Engine and Megascans at GDC*. 2019. Disponível em: <https://www.unrealengine.com/en-US/blog/quixel-demonstrates-stunning-photorealism-using-unreal-engine-and-megascans-at-gdc>. Acesso em: 07 nov. 2019. Citado na página 24.

FIALA, M. Artag, a fiducial marker system using digital techniques. In: . [S.l.: s.n.], 2005. v. 2, p. 590 – 596 vol. 2. ISBN 0-7695-2372-2. Citado na página 21.

Fischer, J.; Bartz, D.; Straber, W. Stylized augmented reality for improved immersion. In: *IEEE Proceedings. VR 2005. Virtual Reality, 2005.* [S.l.: s.n.], 2005. p. 195–202. Citado na página 24.

Fischer, J.; Bartz, D.; Straßer, W. Enhanced visual realism by incorporating camera image effects. *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*, 2006. Citado 2 vezes nas páginas 11 e 15.

FRANKE, T. Delta light propagation volumes for mixed reality. *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013. Citado 2 vezes nas páginas 12 e 13.

FRANKE, T. Delta voxel cone tracing. *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014. Citado 4 vezes nas páginas 10, 12, 13 e 14.

FRANKE, T. *Dirtchamber*. 2017. Disponível em: <https://github.com/thefranke/dirtchamber>. Acesso em: 24 set. 2019. Citado na página 12.

GOOGLE Play. 2019. Disponível em: <https://play.google.com/store>. Acesso em: 11 out. 2019. Citado na página 17.

GOTTSCHALK, S.; HUGHES, J. F. Autocalibration for virtual environments tracking hardware. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1993. (SIGGRAPH '93), p. 65–72. ISBN 0-89791-601-8. Disponível em: <http://doi.acm.org/10.1145/166117.166124>. Citado na página 22.

HIRZER, M. Other, *Marker Detection for Augmented Reality Applications*. 2008. Citado na página 21.

INTEL. *Virtual Reality Vs. Augmented Reality Vs. Mixed Reality*. 2019. Disponível em: <https://www.intel.com/content/www/us/en/tech-tips-and-tricks/virtual-reality-vs-augmented-reality.html>. Acesso em: 15 out. 2019. Citado na página 18.

IZADI, S. et al. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, 2011. (UIST '11), p. 559–568. ISBN 978-1-4503-0716-1. Disponível em: <http://doi.acm.org/10.1145/2047196.2047270>. Citado 2 vezes nas páginas 22 e 23.

KAN, P.; KAUFMANN, H. High-quality reflections, refractions, and caustics in augmented reality and their contribution to visual coherence. In: *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. Washington, DC, USA: IEEE Computer Society, 2012. (ISMAR '12), p. 99–108. ISBN 978-1-4673-4660-3. Disponível em: <https://doi.org/10.1109/ISMAR.2012.6402546>. Citado na página 13.

KAN, P.; KAUFMANN, H. Differential irradiance caching for fast high-quality light transport between virtual and real worlds. In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. [S.l.: s.n.], 2013. p. 133–141. Citado na página 13.

Kato, H.; Billinghurst, M. Marker tracking and hmd calibration for a video-based augmentedreality conferencing system. In: . [S.l.: s.n.], 1999. p. 85–94. ISBN 0-7695-0359-4. Citado na página 21.

KLEIN, G.; MURRAY, D. W. Simulating low-cost cameras for augmented reality compositing. *IEEE Transactions on Visualization and Computer Graphics*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 16, n. 3, p. 369–380, maio 2010. ISSN 1077-2626. Disponível em: <https://doi.org/10.1109/TVCG.2009.210>. Citado 2 vezes nas páginas 13 e 16.

KRONANDER, J. et al. Photorealistic rendering of mixed reality scenes. *Comput. Graph. Forum*, The Eurographs Association & John Wiley & Sons, Ltd., Chichester, UK, v. 34, n. 2, p. 643–665, maio 2015. ISSN 0167-7055. Disponível em: <https://doi.org/10.1111/cgf.12591>. Citado na página 13.

KUNG Fu Panda. Direção de Mark Osborne, John Stevenson. Produção de Melissa Cobb. Roteiro: Jonathan Aibel, Glenn Berger. Glendale: Dreamworks Animation, 2008. Son., color. Legendado.

Kán, P.; Kaufmann, H. Differential irradiance caching for fast high-quality light transport between virtual and real worlds. *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013. Citado na página 11.

LIMA, J. P. et al. Markerless tracking system for augmented reality in the automotive industry. *Expert Systems with Applications*, v. 82, p. 100–114, 10 2017. Citado na página 22.

LOWE, D. Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, v. 2, 01 2001. Citado na página 22.

MAGIC LEAP. *Magic Leap One: Creator Edition*. 2019. Disponível em: <https://www.magicleap.com/magic-leap-one>. Acesso em: 06 nov. 2019. Citado na página 18.

MARR, B. *The Important Difference Between Virtual Reality, Augmented Reality and Mixed Reality*. 2019. Disponível em: <https://www.forbes.com/sites/bernardmarr/2019/07/19/the-important-difference-between-virtual-reality-augmented-reality-and-mixed-reality/#2bf9d09935d3>. Acesso em: 15 out. 2019. Citado na página 18.

MATHWORKS. *Affine Transformation*. 2019. Disponível em: <https://www.mathworks.com/discovery/affine-transformation.html>. Acesso em: 12 dez. 2019. Citado na página 28.

Mei, C.; Reid, I. Modeling and generating complex motion blur for real-time tracking. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008. Citado 2 vezes nas páginas 10 e 15.

MICROSOFT. *Kinect para Windows*. 2010. Disponível em: <https://developer.microsoft.com/pt-br/windows/kinect>. Acesso em: 22 out. 2019. Citado na página 22.

MICROSOFT. *HoloLens2*. 2019. Disponível em: <https://www.microsoft.com/en-us/hololens>. Acesso em: 16 out. 2019. Citado 2 vezes nas páginas 18 e 19.

Milgram, P.; Kishino, F. A taxonomy of mixed reality visual displays. *IEICE Trans. Information Systems*, vol. E77-D, no. 12, p. 1321–1329, 12 1994. Citado 2 vezes nas páginas 17 e 18.

OBSERVATóRIO DO CINEMA. *Especialista revela segredo por trás dos efeitos visuais de Game of Thrones e Vingadores: Ultimato*. 2019. Disponível em: <https://observatoriodocinema.bol.uol.com.br/filmes/2019/04/especialista-revela-segredo-de-efeitos-3d-por-tras-de-game-of-thrones-e-vingadores-ultimato>. Acesso em: 30 abr. 2019. Citado na página 10.

OKUMURA, B.; KANBARA, M.; YOKOYA, N. Augmented reality based on estimation of defocusing and motion blurring from captured images. In: *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*. Washington, DC, USA: IEEE Computer Society, 2006. (ISMAR '06), p. 219–225. ISBN 1-4244-0650-1. Disponível em: <https://doi.org/10.1109/ISMAR.2006.297817>. Citado na página 14.

Park, Y.; Lepetit, V.; Woo, W. Handling motion-blur in 3d tracking and rendering for augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, v. 18, p. 1449–1459, 2012. Citado 6 vezes nas páginas 10, 14, 15, 29, 32 e 46.

PINTARIC, T.; KAUFMANN, H. Affordable infrared-optical pose-tracking for virtual and augmented reality. *IEEE VR Workshop on Trends and Issues in Tracking for Virtual Environments*, 01 2007. Citado na página 22.

PRADA, R. *O que é Motion Blur?* 2008. Disponível em: <https://www.tecmundo.com.br/video-game-e-jogos/724-o-que-e-motion-blur-.htm>. Acesso em: 14 nov. 2019. Citado na página 27.

PTC. *Vuforia Developer Portal*. 2019. Disponível em: <https://developer.vuforia.com/>. Acesso em: 29 out. 2019. Citado 2 vezes nas páginas 29 e 31.

REGENBRECHT, H. et al. Mixed voxel reality: Presence and embodiment in low fidelity, visually coherent, mixed reality environments. In: . [S.l.: s.n.], 2017. p. 90–99. Citado 2 vezes nas páginas 13 e 24.

RIBO, M.; PINZ, A.; FUHRMANN, A. A new optical tracking system for virtual and augmented reality applications. In: . [S.l.: s.n.], 2001. v. 3, p. 1932 – 1936 vol.3. ISBN 0-7803-6646-8. Citado na página 22.

Rohme, K. et al. Interactive near-field illumination for photorealistic augmented reality on mobile devices. *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014. Citado na página 10.

SQUARE ENIX CO., LTD. *FINAL FANTASY 30th ANNIVERSARY EXHIBITION □Memories of You□*. 2017. Disponível em: <http://www.finalfantasy.jp/30th/exhibition/en/>. Acesso em: 13 out. 2019. Citado na página 17.

STANFORD COMPUTER GRAPHICS LABORATORY. *The Stanford 3D Scanning Repository*. 2014. Disponível em: <https://graphics.stanford.edu/data/3Dscanrep/>. Acesso em: 11 dez. 2019. Citado na página 41.

STATT, N. *Pokémon Go iOS update will deliver more advanced augmented reality thanks to ARKit*. 2017. Disponível em: <https://www.theverge.com/2017/12/20/16798388/pokemon-go-niantic-apple-iphone-ios-arkit-augmented-reality-ar-plus>. Acesso em: 17 out. 2019. Citado na página 20.

STUART, K. *Photorealism - the future of video game visuals*. 2015. Disponível em: <https://www.theguardian.com/technology/2015/feb/12/future-of-video-gaming-visuals-nvidia-rendering>. Acesso em: 07 nov. 2019. Citado na página 24.

Ternier, S. et al. Ar learn: Augmented reality meets augmented virtuality. *Journal of Cheminformatics - J Cheminf*, v. 18, 01 2012. Citado na página 18.

THE TELEGRAPH. *The 15 most beautiful video games*. 2011. Disponível em: <https://www.telegraph.co.uk/gaming/what-to-play/the-15-most-beautiful-video-games/>. Acesso em: 13 nov. 2019. Citado na página 25.

THE VERGE. *Minecraft Earth goes a step beyond Pokémon GO to cover the world in blocks*. 2019. Disponível em: <https://www.theverge.com/2019/5/17/18627341/minecraft-earth-ios-android-free-ar-game-features-pokemon-go>. Acesso em: 20 set. 2019. Citado na página 10.

TOFTEDAHL, M.; ENGSTRÖM, H. A taxonomy of game engines and the tools that drive the industry. In: *DiGRA '19 - Proceedings of the 2019 DiGRA International Conference : Game, Play and the Emerging Ludo-Mix*. [s.n.], 2019. Disponível em: <http://www.digra.org/digital-library/publications/a-taxonomy-of-game-engines-and-the-tools-that-drive-the-industry/>. Citado na página 12.

UNITY TECHNOLOGIES. *Dragon Statue*. 2016. Disponível em: <https://assetstore.unity.com/packages/3d/dragon-statue-59053>. Acesso em: 11 dez. 2019. Citado na página 41.

UNITY TECHNOLOGIES. *Rendering and Shading*. 2019. Disponível em: <https://learn.unity.com/tutorial/rendering-and-shading>. Acesso em: 30 out. 2019. Citado na página 31.

UNITY TECHNOLOGIES. *Unity*. 2019. Disponível em: <https://unity.com>. Acesso em: 29 out. 2019. Citado 2 vezes nas páginas 29 e 30.

WIKIART VISUAL ART ENCYCLOPEDIA. *Gumball II*. 2012. Disponível em: <https://www.wikiart.org/en/charles-bell/gumball-ii-1973>. Acesso em: 06 nov. 2019. Citado na página 24.

Wrzesien, M. et al. The therapeutic lamp:treating small-animal phobias. *IEEE Computer Graphics and Applications*, v. 33, 01 2013. Citado na página 19.

Yeon Ma, J.; Soo Choi, J. The virtuality and reality of augmented reality. *Journal of Multimedia*, v. 2, 02 2007. Citado na página 18.

ZHANG, Z. Microsoft kinect sensor and its effect. *IEEE Multimedia - IEEEMM*, v. 19, p. 4–10, 02 2012. Citado na página 22.