Desenvolvimento rápido de ambientes para realidade virtual em Unity utilizando PhotoSphere e CubeMap

Ricardo da Silva Barboza* Matheus Palheta Barbosa Jucimar Maia da Silva Jr.

Universidade do Estado do Amazonas, Escola Superior de Tecnologia, Brasil

RESUMO

Este trabalho apresenta um histórico sobre realidade virtual (VR), o atual estado da tecnologia e do mercado, ressaltando a importância do ambiente na experiência em VR. Em seguida, apresenta uma técnica de desenvolvimento de ambiente virtual usando *Photosphere* ou *Cubemap*, em Unity.

Palavras-chave: vr, realidade virtual, unity, gear vr, cardboard.

1 INTRODUÇÃO

Realidade Virtual é um conjunto de tecnologias cujo objetivo é sequestrar nossos sentidos da realidade, oferecendo-lhes estímulos simulados que são interpretados como reais pelo nosso cérebro.

Tecnologias como *displays* estereoscópicos, rastreamento de movimento, simuladores de movimento e áudio espacial, são utilizados em dispositivos de realidade virtual para simular estímulos que são determinados por um software, enganado nossos principais sentidos como audição, visão e tato. Essas tecnologias combinadas são a base dos dispositivos de realidade virtual atualmente, porém novos dispositivos/tecnologias podem melhorar a experiência do usuário de VR.

A realidade virtual sempre chamou atenção pelas possibilidades que ela cria, de novas formas de interação com o ambiente virtual, à maior possibilidade de impacto emocional em experiências virtuais, devido a maior imersão que essa tecnologia é capaz de causar no usuário [1].

2 HISTÓRICO

Estudos acadêmicos sobre realidade virtual tem sido realizados desde os anos 50, com dispositivos como o *Sensorama* de Morton Heilig (Figura 1), e a *sword of damocles* de Ivan Sutherland (Figura 2), considerado o primeiro *Head Mounted Display* (HMD – Tela acoplada a cabeça) e a base para os dispositivos de realidade virtual que estão no mercado atualmente [2].

Desde então dispositivos de VR vem sendo desenvolvidos e utilizados em simulações, como por exemplo:

1. *VCASS* (*Visually Coupled Airborne Systems Simulator*) HMD para simulação de voo, criado em 1982 por Thomas Furness na Força Aérea Americana (Figura 3).
2. *VIVED* (*Virtual Visual Environment Display*), criado pela *NASA* em 1984, possuía um HMD estereoscópico monocromático (Figura 4).

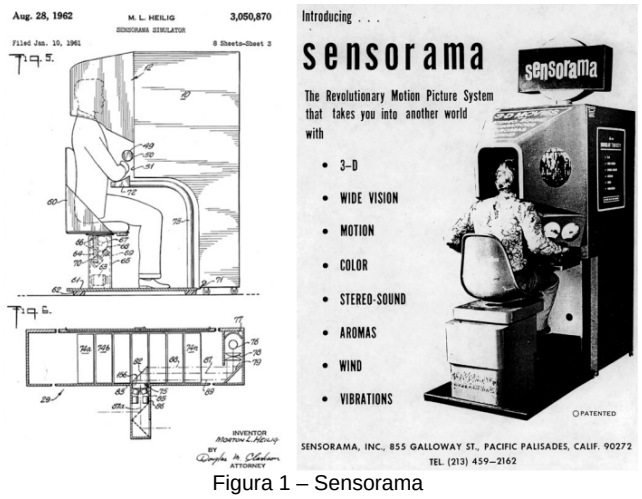

Figura 1 – Sensorama

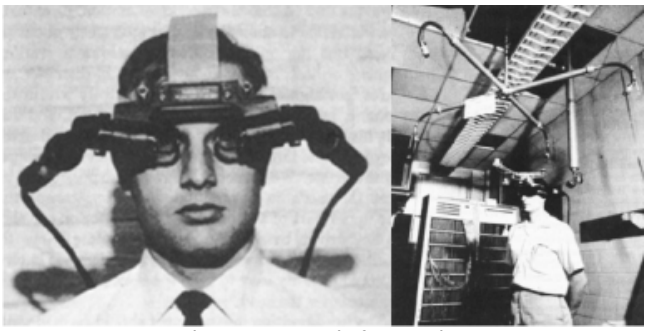

Figura 2 – Sword of Damocles

3. *BOOM* (*Binocular Omni-Orientation Monitor*), desenvolvido em 1989 pela *Fake Space Labs*, dispositivo HMD com dois monitores CRT acoplados numa pequena caixa (Figura 5).
4. *Virtual Wind Tunnel*, criado em 1990 pela NASA, foi criado com o objetivo de visualizar e testar fluxos de ar em simulações 3d, usando o dispositivo HMD e um Dispositivo de *Hand Tracking* chamado *DataGlove* (Figura 6).
5. *CAVE* (*Cave Automatic Virtual Environment*), inventado em 1992, é um sistema de visualização científica e VR, consiste de uma sala com paredes de LCD onde o usuário precisa de um *Shutter Glass* (Óculos próprios para visualização 3D) (Figura 7) [15].

*e-mail: rsbarboza@gmail.com

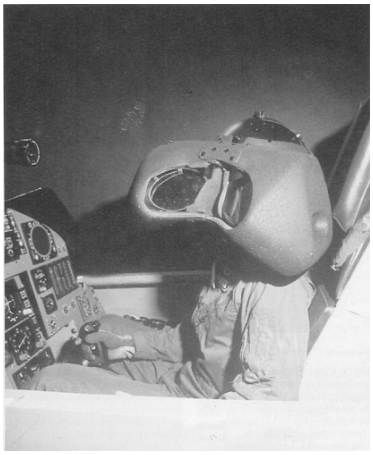

Figura 3 – Visually Coupled Airborne Systems Simulator

Figura 4 – Virtual Visual Environment Display

Figura 5 - Binocular Omni-Orientation Monitor

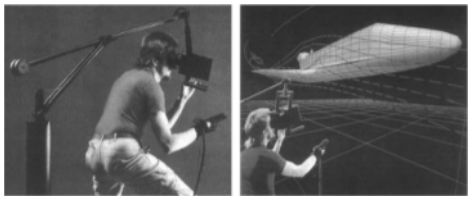

Figura 6 - Virtual WindTunnel

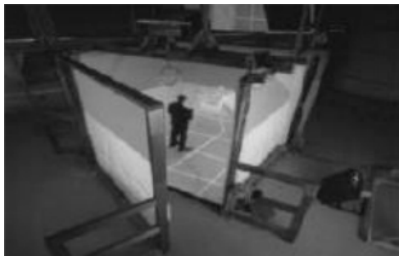

Figura 7 - Cave Automatic Virtual Environment

Nos anos 90, houve um crescimento no interesse em dispositivos de VR, principalmente pela indústria de jogos. Algumas tentativas de produtos comerciais foram postas no mercado, como o *Sega VR* (Figura 8) em 1991 e o *Nintendo Virtual Boy* (Figura 9) em 1995, e diversas máquinas de *arcade* ofereciam experiências em VR, como por exemplo o *virtuality* (Figura 10) porém com resultados insatisfatórios. Diversos problemas na experiência, como gráficos de baixa fidelidade, falta de dispositivos de interação e baixa taxa de quadros por segundo, causavam desconforto e náusea nos jogadores, o que afastou o público dessa tecnologia [4].

Figura 8 - Sega VR

Figura 9 – Nintendo Virtual Boy

Figura 10 - Virtuality

3 DISPOSITIVOS DE REALIDADE VIRTUAL

O interesse no mercado de VR teve um ressurgimento com o Oculus Rift. Com uma campanha de financiamento coletivo de sucesso e uma avaliação positiva por parte dos primeiros usuários [6], o sucesso do Oculus Rift demonstrou que ainda existia interesse do mercado em dispositivos e experiências em VR, então diversas empresas começaram a desenvolver seus dispositivos, como a HTC e a Valve com o HTC Vive, a Sony com o Playstation VR, a Google com o Cardboard e o Daydream, para plataformas móveis, e a Samsung em parceria com a Oculus, com o Gear VR, também para plataformas móveis [5]. Essas são as principais plataformas de VR atualmente, com cerca 6,18 milhões de dispositivos vendidos apenas em 2016. Desses, cerca de 4,77 milhões são dispositivos para plataformas móveis, como o Gear VR e o Google Daydream (Tabela 1) [7].

A plataforma para qual se desenvolve determina o tipo de experiência que é possibilitada ao usuário [1]. Oculus Rift (Figura 10), HTC Vive (Figura 11) e Playstation VR (Figura 12) possuem controles de *tracking* de movimento das mãos, possibilitando uma interação mais intuitiva do usuário com o ambiente em VR, podendo usar o movimento da mão do usuário para ele possa interagir com os objetos do ambiente. Já dispositivos como o Google Cardboard (Figura 14) ainda não possuem nenhum controle de *tracking* de movimento das mãos, o que limita um pouco a forma como o usuário pode interagir com o ambiente virtual. O GearVR (Figura 13) possui um *touchpad* na lateral do dispositivo, onde o usuário pode interagir com comandos de toque e *swipes.* Além disso, o GearVR possui um dispositivo de *tracking* de movimento, possibilitando mais interações do usuário com o ambiente virtual[4].

Figura 10 - Oculus Rift

Figura 11 – HTC Vive

Figura 12 – Playstation VR

Figura 13 – Gear VR

Figura 14 – Google Cardboard

Em plataformas VR *mobile*, utiliza-se principalmente a interação através do olhar (*gaze*), onde o usuário tem uma mira que ele usa para olhar para os objetos no ambiente virtual e interagir com estes [8]. Apesar do *touchpad* e do controle de movimento, a maioria das experiências para GearVR permite que o usuário interaja apenas com olhar, uma escolha do desenvolvedor para tornar a experiência mais abrangente, sem necessidade de dispositivos auxiliares, e prontas para funcionar em dispositivos sem esse tipo de interação, como o Google Cardboard [10].

Pode-se tomar como exemplo de interação *gaze* da Oculus Home (Figura 15) no menu principal do Gear VR onde o usuário pode acessar e baixar outros aplicativos.

Figura 15 – Oculus Home do Gear VR

No Oculus Home apresentam-se a mira na tela indicando onde o usuário está olhando, com qual objeto ele vai interagir, uma interface com as opções do usuário, e ao redor pode-se ver um ambiente de fundo, nesse caso, uma sala de estar aconchegante. A interação do usuário com a interface já é conhecida e estudada em aplicativos em 2 dimensões, porém em VR acrescenta-se um novo elemento: imersão. O usuário estará imerso num ambiente, e esse ambiente precisa ser interessante mesmo que ele não faça parte da interface, mas faz parte da experiência que o usuário vai ter.

4 MERCADO DE REALIDADE VIRTUAL

Desde que a Oculus foi comprada pelo Facebook, em março de 2014, por US$ 2 bilhões, houve um aumento nos investimentos em VR [6]. Quase no mesmo período a Sony anunciava o seu dispositivo de VR, *Project Morpheus*, depois renomeado *Playstation VR*, e lançado em outubro de 2016. Em junho, O Google anunciou o Google Cardboard, plataforma VR de baixo custo que utiliza um *smartphone*. Em 2014, em setembro, a Samsung anunciou o Samsung Gear VR, em parceria com a Oculus, que também utiliza um *smartphone* como dispositivo, lançado em novembro de 2015, foi a plataforma VR mais adotada pelos usuários, com 4,5 milhões de dispositivos enviados ao consumidor (Tabela 1). Em março de 2015, a HTC anuncia o HTC Vive, em parceria com a Valve, que foi lançado em abril de 2016 [6].

Tabela 1 – Quantidade de dispositivos VR vendidos no ano de 2016 [7].

Dispositivo	Quantidade (em milhões)
Oculus Rift	0,24
HTC Vive	0,42
Playstation VR	0,75
Samsung Gear VR	4,51
Google Daydream	0,26
Total	**6,18**

Como se pode observar, as plataformas *mobiles* são as mais adotadas pelo consumidor, enquanto VR para *desktop* está sendo mais usado por empresas [6]. Isso se deve a baixa barreira de entrada dos dispositivos móveis, tanto em relação ao preço, quanto à comodidade e facilidade [7].

Os investimentos na área de VR/AR chegaram a US$ 3,5 bilhões até 2016, incluindo a aquisição da Oculus pelo Facebook, investimentos da Google, Intel, Apple, tanto em AR quanto em VR.

Algumas barreiras ainda dificultam a adesão de dispositivos HMD, como o preço de um *hardware desktop* capaz de rodar o Oculus Rift/HTC Vive, em torno de US$ 1.500. Pesquisas revelaram que em 2016, quando esses dispositivos foram lançados, apenas 1% da base instalada de computadores eram capazes de rodar VR [7].

Sendo assim, acredita-se que as plataformas móveis sejam as impulsionadoras do VR para o público geral como forma de entretenimento, sendo mais econômicas e práticos, enquanto experiências em *desktops* sejam utilizados por empresas para fins de interação e praticidade [5].

Estimativas apontam que o mercado de VR/AR alcance o valor de US$ 80 bilhões em 2025, levando em conta a evolução da tecnologia, a redução de preço dos dispositivos, e consequentemente a maior adoção dos usuários, principalmente em plataformas *mobile* [5]. Aponta-se como principais mercados a adotar VR/AR em seus modelos de negócios:

1. Videogames: O mercado de jogos vem explorando a possibilidade da VR desde os anos 90 [4], porém com as novas e melhores tecnologias a possibilidade de jogos atraentes ao publico tornou-se realidade, com diversos jogos fazendo sucesso no Playstation VR e na Steam. E acredita-se que quanto mais jogos forem sendo aprovados pelo público, mais pessoas vão se interessar, aumentando a base de usuários [6].
2. Transmissão de eventos: A transmissão de eventos em 360 usando VR já tem alguns experimentos [10].
3. Vídeos: Diversos aplicativos para visualização de vídeos em 360 estão disponíveis na Oculus Store [10], incluindo Youtube, Discovery e *Cirque du soleil*.
4. Imobiliária: A possibilidade de mostrar um imóvel a um comprador em VR tem atraído a atenção de empresas e corretores [6].
5. Educação: A possibilidade de imersão e interação em um ambiente 3d podem tornar mais interessantes e didáticos certos assuntos.
6. Saúde: Na área de saúde a VR pode ajudar muito na parte de ensino/treinamento, e também na acessibilidade a atendimento médico, usando VR [6].
7. Engenharia: A VR pode ajudar engenheiros a projetar e testar produtos e ferramentas, usando o ambiente virtual.
8. Militar: O setor militar vem usando simulações e VR a bastante tempo, inclusive ajudando no desenvolvimento dessa área de pesquisa.

5 EXPERIÊNCIAS EM VR

Alguns aspectos de hardware diferenciam os dispositivos VR para *desktop* e *mobile*. Enquanto hardware *mobile* é consideravelmente inferior ao *desktop* em questão de processamento, pode-se citar como vantagens: Mobilidade, praticidade e economia. As diferenças de hardware com o tempo vão sendo diminuídas, e cada vez mais são criados dispositivos que melhorem a interação do usuário com o ambiente virtual em *mobile* [2].

Desenvolvimento de aplicativos para *mobile* deve levar em conta as limitações de processamento e interação. Apesar disso, diversos tipos de experiências VR podem ser desenvolvidos mesmo levando em conta essas limitações, a saber [1]:

1. Videogames: Apesar das limitações das entradas, é possível criar jogos com jogabilidades simples, utilizando do VR para ampliar a sensação de imersão do jogo, ou criar experiências mais similares a jogos tradicionais utilizando um *gamepad*.
2. *Virtual worlds*: uma experiência que se encaixaria como algo entre jogos e simuladores, mundos virtuais seriam as redes sociais das plataformas em realidade virtual.
3. Educação: Com a ajuda de aplicativos de realidade virtual é possível fazer melhores demonstrações e explicações de assuntos didáticos, a interação e imersão providos pelo ambiente virtual tornam o ensino mais didático e eficiente.
4. Produtividade: A realidade virtual pode servir para substituir as mesas de trabalho, criando um ambiente 3D funcional com uma organização do espaço de trabalho mais eficiente.
5. Turismo, arquitetura, eventos: pode-se utilizar a realidade aumentada para criar aplicações de telepresença, utilizando fotos e vídeos de um outro ambiente, o usuário é capaz de imergir em um outro local, explorá-lo e ter uma noção da ambientação.
6. Navegação web: existem algumas experiências de como seria a navegação na web utilizando VR, como o Chrome Beta (M56+), Firefox Nightly e o Samsung Internet do Gear VR.

7. Simulações: Com os dispositivos e tecnologias certas é possível criar simulações para os mais diversos fins: Médicos, militares, aviação, automobilístico, etc.

Em dispositivos mobile é necessário levar em conta as limitações do aparelho na hora de desenvolver um projeto em VR, o tempo de duração médio da experiência e quais tipos de periféricos o aparelho suporta.

6 DESENVOLVIMENTO PARA VR MOBILE

Com o crescente interesse, diversas ferramentas de desenvolvimento foram criadas para trabalhar com VR, e especialmente, diversas plataformas de desenvolvimento de jogos agora suportam o desenvolvimento para VR.

A principal *engine* (motor gráfico) de desenvolvimento de jogos multiplataforma, Unity, suporta desenvolvimento de experiências para Cardboard, GearVR, Oculus Rift, HTC Vive e Playstation VR.

A *engine* de desenvolvimento Unreal também possui suporte ao Gear VR, Google Daydream, Oculus Rift e HTC Vive.

Essas ferramentas permitem o desenvolvimento de aplicações em VR para dispositivos mobile, mas também é possível utilizar desenvolvimento nativo na plataforma alvo (Android/Ios).

Para o desenvolvimento de aplicativos em VR para dispositivos mobile, como o Gear VR ou Google Cardboard, deve se levar em consideração as limitações de performance e sensores desses dispositivos.

A falta de controles é um dos principais problemas dessas plataformas [4], limitando a interação do usuário e o tipo de experiência que o desenvolvedor pode passar. O *head tracking* desses dispositivos permite que desenvolvamos uma interação baseada em *Gaze Input* [8], onde o usuário usa o olhar para selecionar e interagir com objetos virtuais.

Dispositivos *smartphone* permitem também que sejam usados *gamepads* próprios para mobile, porém é recomendado a utilização de *gaze input* para não limitar o alcance de usuários do aplicativo [8].

Outro aspecto limitante dos dispositivos de VR mobile é a capacidade de processamento desses dispositivos, sendo consideravelmente inferiores em relação aos dispositivos *desktop*/consoles [4], o que reduz a capacidade do desenvolvedor de explorar gráficos de alta-fidelidade na experiência VR, e outras técnicas que demandem muito processamento. Segundo os *guidelines* da Oculus [9], um aplicativo VR deve manter uma taxa de quadros (*fps*) estável, de preferência a 60 fps, para manter o movimento da câmera no ambiente correspondente ao movimento de cabeça do usuário, evitando desconforto. Para isso, recomenda-se uma série de restrições no desenvolvimento como limitar a quantidade de polígonos e *draw calls* necessários na cena, recomenda-se utilizar apenas uma câmera e não utilizar *post-processing effects*, além de recomendar o uso de imagens panorâmicas (360) para o ambiente [9].

Lembrando que mais cuidados devem ser tomados ao desenvolver para Cardboard, levando em consideração a variedade de dispositivos *smartphone* que são capazes de rodar aplicativos em Cardboard, alguns com muito menos capacidade de processamento dos que são compatíveis com o Gear VR [8]. Além do problema da manutenção da taxa de quadros, *smartphones* rodando aplicações em VR podem sofrer de superaquecimento, que pode causar gasto excessivo da bateria do aparelho, reduzindo a vida útil do aparelho, e problemas relacionados ao aquecimento dos componentes do dispositivo [9], além de ser perigoso para o usuário o superaquecimento do aparelho próximo de sua face.

Devido a isso, aplicativos em VR devem ser desenvolvidos levando em conta a otimização como um dos principais objetivos, para entregar ao usuário uma experiência não só engajante e imersiva, mas que seja agradável.

7 AMBIENTE EM VR

Um aspecto muito importante da imersão em um aplicativo de VR é o ambiente simulado que o usuário se encontra [1]. Um ambiente de realidade virtual funcional, com interface agradável, porém com uma ambientação simples (um fundo de uma cor, por exemplo), pode diminuir a satisfação do usuário no uso desse aplicativo.

Os aplicativos de VR costumam criar ambientes imersivos, aconchegantes e que estejam de acordo com a temática e funcionalidade do mesmo.

Tomando como exemplo alguns aplicativos da Oculus Store [9]:

1. Oculus Home, Oculus Rooms, Oculus Avatar Editor: Os aplicativos principais do Oculus simulam um ambiente de uma casa, sendo o Oculus Home, onde o usuário acessa suas configurações, aplicativos, etc., uma sala de estar; o Oculus Rooms, onde o usuário pode se encontrar virtualmente e interagir com outros usuários, simula uma sacada; e o Oculus Avatar Editor, aplicativo da Oculus para edição do avatar virtual do usuário, simula um armário de roupa.
2. Pigasus Streaming Theater: Aplicativo para visualização de vídeos em 2D e 3D, ele simula um ambiente de uma sala de cinema.
3. O aplicativo Speech Center VR é um aplicativo cujo objetivo é treinar pessoas para situações em que seja necessário falar em publico, para isso ele simula diversas situações, como aulas, palestras e conversas.
4. Relaxating VR Games: Mahjong é um jogo de *mahjong*, porém em VR ele criar uma ambientação oriental com objetivo de ser uma experiência relaxante.
5. No aplicativo do Facebook para VR, o ambiente principal é um parque.
6. Oculus Arcade permite que você jogue em *arcades* virtuais, diversos jogos da Sega, Midway e Namco, num ambiente que lembra antigas casas de *arcade*.
7. O aplicativo Gear 360 London, um aplicativo de visualização de fotos, possui como ambiente principal um terraço de prédio, onde é possível acessa as outras fotos, como se fossem localizações na cidade.
8. The World Cup of Hip Hop é um aplicativo com diversos vídeos contando a história de alguns *rappers*, e o ambiente principal é uma mesa de DJ numa festa.
9. Disney Movies VR é um aplicativo com diversos vídeos 360 produzidos pela Disney para promover seus filmes, o menu principal da aplicação possui um ambiente bem interessante, com o castelo da Disney, a torre dos Vingadores e a estrela da morte de Star Wars.

Estas aplicações exemplificam a importância da ambientação para a melhor imersão do usuário na realidade virtual. A partir delas pode-se observar alguns padrões que são seguidos no desenvolvimento de ambientes virtuais, como evitar ambientes com poucos detalhes, com apenas paredes coloridas, colocar elementos que remetam a funcionalidade do aplicativo, como uma sala de cinema para vídeos ou uma sala de estar para conversas.

8 INTERAÇÃO EM VR MOBILE

Devido às limitações de entrada (*input*), os *guidelines* de desenvolvimento tanto do Oculus [9] quando do Cardboard [8] definem alguns padrões, como o uso de uma mira para que o usuário possa interagir com a interface, usando também um temporizador (*timer*) com indicador visual para informar o usuário o estado da interação [8].

Em relação a interface, a documentação do Oculus [9] cita o uso de HUDs (*heads-up display* - tela de alerta) em jogos, e como esse tipo de interface não é recomendada em aplicativos em VR, colocar objetos fixos a visão do jogador pode tanto interferir com a noção de imersão dele quanto causar desconforto. As interfaces em VR costumam ser do tipo espaciais, elas são integradas ao ambiente em que se encontram, não apenas uma camada a mais de informação, mas sim informação contida no mundo virtual [12].

É possível usar tanto elementos 2D quanto 3D para criar uma interface para realidade virtual, porém deve-se sempre levar em conta o espaço do ambiente para posicionar esses elementos. Podemos trabalhar com caixas ou esferas selecionáveis (Figura 16), ou utilizar planos no ambiente 3D como telas para posicionar os elementos de interface 2D (Figura 17). A escolha do tipo de interface fica a critério do desenvolvedor, de acordo com a ambientação, o tipo de interação e a experiência que ele quer transmitir.

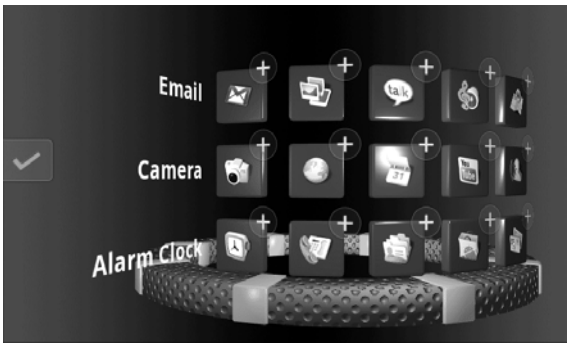

Figura 16 – Interface com elementos 3D

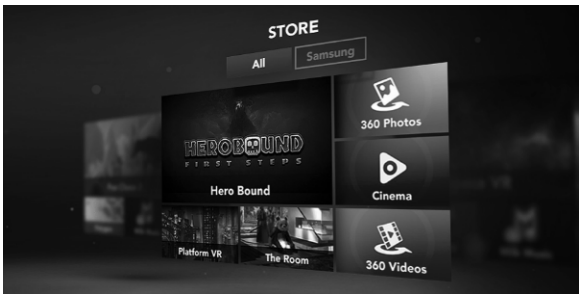

Figura 17 – Interface com planos 2D

Sendo assim, a posição dessa interface no ambiente altera a forma como o usuário interage com ela. Elementos de interface podem ser posicionados ao redor do usuário em relação a importância das ações e informações desses elementos. Assim como numa aplicação *desktop* ou *mobile* os elementos mais importantes são posicionados de maneira que o usuário possa interagir com mais facilidade, no caso do VR, na frente de sua visão. Em um menu principal, coloca-se os principais elementos diretamente em frente ao usuário, distribui-se os outros elementos ao redor, e evita-se colocar coisas atrás do usuário. Se possível, posicione os elementos numa interface circular, ou então com uma quantidade de planos ao redor do usuário, levando sempre em conta que ele deve ter visão plena de todos os elementos da interface, evitando que objetos fiquem em uma angulação em relação a visão do usuário que dificulte a visualização dos mesmos.

Os elementos de interface devem ser distribuídos de forma que a interação a partir da mira em um, não interfira em outro elemento, ou seja, não haja obstrução de um elemento sobre outro, ou que eles estejam muito próximos a ponto do usuário selecionar um elemento que ele não queria. Distribua os objetos pela interface de forma que evite esse tipo de problema [12].

Mantenha a tela a uma distância confortável do usuário. Podendo causar dificuldades no usuário em focalizar nesses objetos. Objetos muito longe também não são recomendados, a partir de uma certa distância não é possível perceber diferença de 3D. Em Unity, se recomenda posicionar objetos entre 0,5 e 20 unidades de distância [9].

Mantenha os objetos direcionados para a câmera, assim fica mais fácil para o usuário visualizá-los. Muitos aplicativos fazem com que os elementos de interface sempre direcionem para o usuário, movendo-os conforme a rotação da câmera.

9 PROCESSO DE DESENVOLVIMENTO PARA VR

Como a experiência de um aplicativo em Realidade Virtual tem muita semelhança com um jogo em 3D, o processo de desenvolvimento para VR tem uma grande semelhança com o processo de desenvolvimento de um jogo (Figura 18).

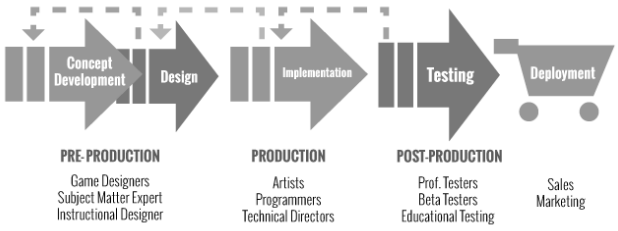

Figura 18 – Processo de desenvolvimento de um jogo [14]

Em um processo de desenvolvimento de jogo, é necessário pelo menos alguns papéis num time básico de desenvolvimento:

1. *Game Designer*: Responsável pela ideia geral do jogo, mas não necessariamente é o idealizador. Ele define tudo aquilo que concerne a jogabilidade e enredo.
2. *Level Designer*: Responsável por definir as mecânicas de jogo e o layout das fases do jogo. Geralmente seu foco é na diversão e jogabilidade, trabalhando em conjunto com o game designer para encaixar as ideias de jogabilidade num ambiente interessante para o jogador.
3. *Graphic Designer*: Responsável pelo visual, desde a concepção artística até imagens e modelos 3D usados jogo.
4. *Sound Designer*: Responsável pela parte sonora do jogo, música e efeitos.
5. *Game Developer*: Responsável pela parte de codificação do jogo, vai trabalhar com a *engine* para dar vida aos conceitos e visual do projeto.
6. *Tester*: Responsável por verificar o funcionamento e a diversão do jogo.

Alguns outros papéis podem ser adicionados conforme a necessidade (*Story writer*, *Art director*, etc.), e vários papéis podem ser assumidos pela mesma pessoa [14].

Em um processo de ágil de desenvolvimento de jogo, o projeto é divido nas seguintes etapas:

1. Concepção e ideação: O processo de concepção da ideia de um jogo, onde artistas e game designers vão definir jogabilidade, enredo, estilo de arte, clima, etc. Envolvem vários processos de criatividade (*Brainstorm*, *design thinking*, entre outros) para capturar as melhores ideias que podem ser aplicadas no jogo.
2. Projeto: Com o escopo do jogo definido, pode-se projetar a estratégia de desenvolvimento que se ajuste ao projeto. Nesse passo definissem o GDD (*Game Design Document*), *storyboard*, e estipulam-se atividades e *milestones* para viabilizar o projeto em um tempo determinado.
3. Protótipo: Com os elementos do jogo definido, pode-se desenvolver um simples protótipo, tanto para testar a viabilidade quanto o fator diversão do jogo.
4. Implementação: Aqui começa a parte de desenvolvimento, envolvendo todo o time (*designer*, *programadores*, *sound designer*). Dependendo do modelo de projeto estipulam-se *milestones* onde uma versão jogável desenvolvida, com alguma mecânica principal.
5. Teste: Testes em desenvolvimento de jogos envolvem muito mais que software funcional, a diversão do jogo é também avaliada e o interesse do público. Dependendo do modelo de processo, os testes são executados a cada *milestone* alcançado pelo desenvolvimento. É preferível que exista um time especifico para testes, chamado *Quality Assurance* (QA).

Apesar da divisão de papéis, é muito comum o envolvimento de toda a equipe no processo de concepção e ideação, principalmente em equipes pequenas de até 7 pessoas [14].

O processo de desenvolvimento para VR é muito parecido com esse processo com apenas algumas modificações nos papéis e etapas:

1. O *Game designer* dá lugar ao *UX Designer*, responsável por definir a experiência do usuário, como ele vai interagir com o ambiente virtual, o que ele pode ou não fazer, etc.
2. *Level designer* poderia ser comparado com um *UI designer*, responsável por definir a interface com a qual o usuário vai interagir, levando em conta o espaço 3D e a noção espacial do usuário.
3. Modelador 3D: Responsável pelos objetos 3D do ambiente, é um papel importante, já que todo o ambiente que o usuário ira interagir é 3D. Deve também ter o cuidado especial de modelar objetos com baixa contagem de polígonos, de acordo com a plataforma, para atender as exigências de desempenho esperadas pelo usuário.
4. *Software Developer*: Basicamente um *game developer*, porém em vez de trabalhar com mecânicas de jogo, trabalha com as interações definidas pelo *UX designer*.
5. A equipe de testes precisa estar atenta aos detalhes e limitações do dispositivo alvo da aplicação. Testes de usabilidade, de interação, de desempenho, todos são extremamente importantes para garantir que o usuário tenha uma experiência agradável, dada a facilidade com que é possível causar desconforto e náusea no VR.

Sendo assim, o processo de desenvolvimento de software para VR pode ser dividido em:

1. Design de Experiência/Interface: Deve-se levar em conta a natureza particular da realidade virtual, o ambiente 3D. Enquanto numa tela 2D a interface fica posicionada em relação a tela, no ambiente 3D a interface não pode ficar acoplada a visão do usuário, pois isso causa desconforto. Usa-se de interface espaciais, posicionadas no ambiente, para possibilitar interação do usuário com objetos. Também deve se levar em conta o campo de visão do usuário (Figura 19), a forma como ele vai interagir (*Hand Tracking*, *Gaze Input*), o tipo de interface (planos 2D, objetos 3D), e outras diversas diferenças inerentes a experiência VR.

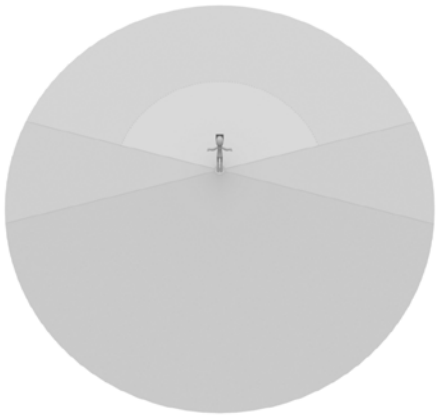

Figura 19 – Campo de visão do usuário em VR

2. Projeto de sistema: Projeto de sistema em VR deve levar em conta as dificuldades no desenvolvimento, como o processo de modelagem 3D, otimização de performance, etc.
3. Desenvolvimento: O processo de desenvolvimento de um aplicativo VR tem muito mais semelhança com o desenvolvimento de jogos ou programas de visualização 3D, as vezes utilizando ferramentas e processos oriundos dessas plataformas, como no caso do Unity e do Blender.
4. Testes: O processo de QA em VR possui algumas particularidades, devido a natureza mais intimista e imersiva de realidade virtual, o aplicativo pode causar desconforto e náusea no usuário, devido a problemas técnicos, como performance, ou de design, como objetos muito próximos do usuário.

Como visto, o desenvolvimento de um aplicativo para VR tem características tanto de desenvolvimento de software padrão, quanto de desenvolvimento de jogos, além de possuir algumas características específicas. Como uma plataforma nova, ainda existe muito a ser experimentado.

Claro que esse processo não é sempre o ideal, dependendo do tipo de experiência pode-se utilizar reduzir ou ampliar o escopo, e consequentemente, o número de processos e papéis envolvidos.

Por exemplo pode-se mitigar a necessidade de um modelador de ambiente 3D, utilizando imagens 360 para criar ambiente pré-renderizados. Essa técnica elimina a necessidade de modelagem 3D, um processo trabalhoso e demorado, e reduz o processamento de modelos 3D, melhorando a performance.

10 AMBIENTE PRÉ-RENDERIZADO

Pode-se usar uma imagem 360 como ambiente pré-renderizado para VR. Ambientes pré-renderizados já são utilizados em jogos desde o início do desenvolvimento em 3D. A ideia de ambientes pré-renderizados é que sejam imagens estáticas que funcionem junto com objetos 3D na cena, diminuindo o número de polígonos necessários para renderizar a cena. Como no desenvolvimento para VR uma das principais preocupações é com a performance, o uso de ambiente pré-renderizado ajuda no alcance do desempenho ideal, além de facilitar o desenvolvimento, reduzindo o trabalho de modelagem 3D.

Em VR, é possível utilizar uma imagem 360 como ambiente. Diversos aplicativos são focados na visualização de fotos e vídeos em 360 que podem ser utilizadas para criar ambientes interessantes para o usuário. Pode-se aproveitar do fato de que a movimentação em VR ainda estar limitada a direção a qual o usuário está observando, ou seja, sem que ele se mova pelo

cenário, para colocá-lo num ambiente pré-renderizado. Como o usuário está sempre preso a um ponto no espaço virtual, é possível renderizar como seria o ponto de vista dele nesse ponto e aplicá-la no ambiente.

Imagens 360 podem ser utilizadas para simular um ambiente. As formas mais comuns para esta composição utilizam *Cubemap* e *Photosphere*.

Um *cubemap* é um conjunto de 6 imagens que representam as projeções de um ambiente a partir de um ponto de vista (Figura 20). As projeções são escolhidas de forma que a junção dessas imagens num cubo forma uma imagem 360 a partir desse ponto de vista, geralmente com um ângulo de visão de 90º, e tomando como um ponto de vista como "frente", define-se os outros ângulos "esquerda", "direita", "encima", "abaixo" e "atrás". É muito utilizado em desenvolvimento de projetos 3D (Jogos e CG) para aplicar reflexão em um objeto.

Photosphere é uma projeção planificada de uma esfera, também chamada de projeção equiretangular, é a projeção utilizada na maioria dos mapa-mundis, projetando o globo terrestre num retângulo (Figura 21).

Figura 20 – Cubemap de um ambiente

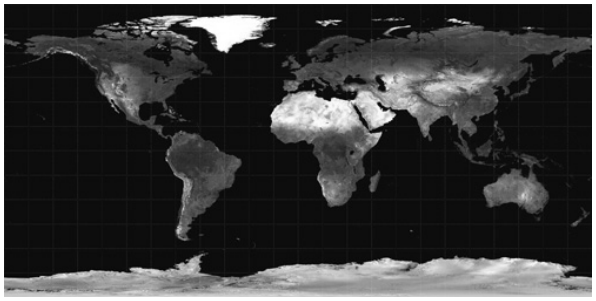

Figura 21 – O mapa-mundi é uma projeção equiretangular da terra

Cubemap é utilizado em diversas ferramentas de 3D e jogos para simular reflexão em objetos, projeção de texturas, *Skyboxes*, entre outros [11].

Photosphere é utilizado o formato de imagem padrão de diversos aplicativos de visualização de fotos em 360, incluindo o Google Street View, Facebook, diversas câmeras 360 como Gear 360 [10].

Uma aplicação interessante de ambiente pré-renderizado em 360 é na criação de ambientes fotorrealistas, em que a renderização em tempo real dessa cena fosse inviável em VR. Pode-se renderizar a cena numa imagem e utilizá-la, sem ter a perda de qualidade que seria sentida com o modelo 3D muito pesado graficamente para ser processado pelo dispositivo.

Para demonstrar esta diferença a Figura 22 compara um ambiente modelado em 3D, e uma imagem 360 renderizada a partir desse modelo.

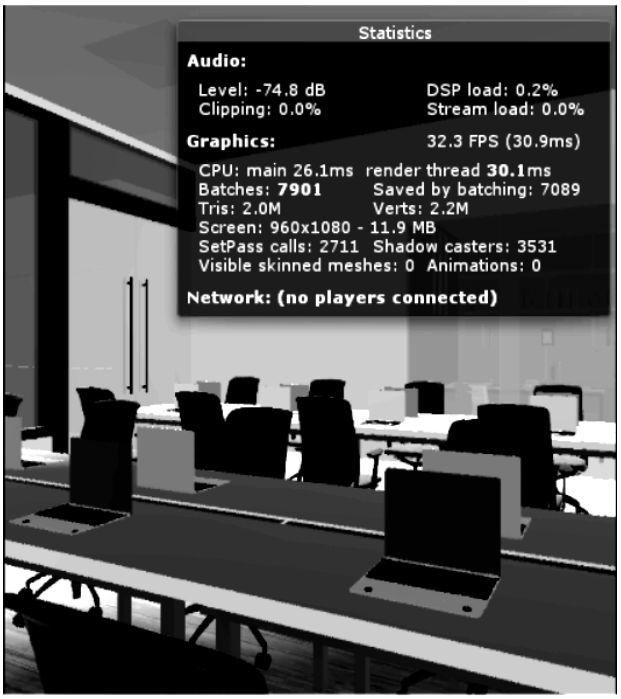

Figura 22 – Informações de performance usando modelagem 3D no Unity.

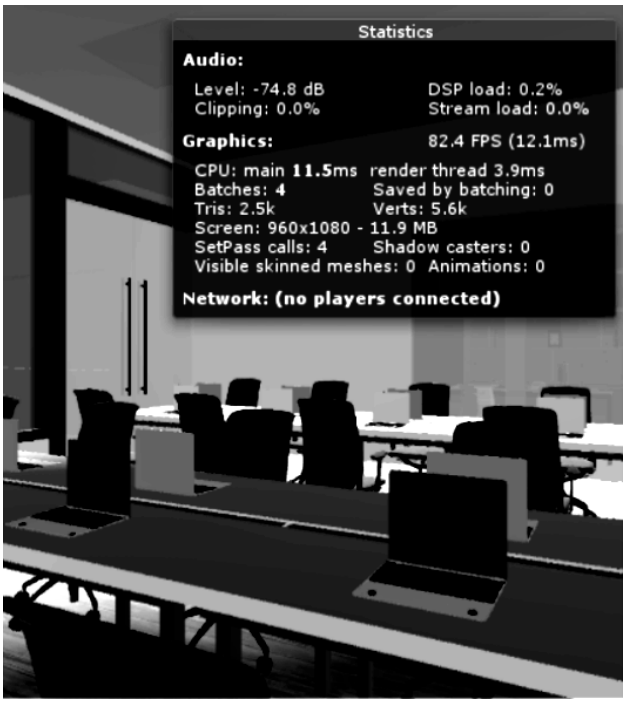

Figura 23 – Informações de performance usando Photosphere no Unity.

Como se pode ver nas Figuras 22 e 23, a utilização um modelo 3D no Unity pode demandar muito processamento e dependendo da quantidade de triângulos, um modelo 3D bem detalhado pode tornar sua utilização em VR inviável por causa da taxa de quadros, no caso em 30 fps (*frames per second*). Já utilizando um *cubemap*, gerado a partir do ponto de vista da câmera nessa cena, como mostrado na Figura 23, o número de triângulos no quadro é bem menor e a renderização demora 12ms, com uma taxa de quadros de 80 fps, ideal para uma experiência VR.

11 DESENVOLVIMENTO DE UM PASSEIO VIRTUAL EM VR

Como estudo de caso, será demonstrado o processo de desenvolvimento de um aplicativo simples para realidade virtual que chamamos OceanVR. Nesse aplicativo o usuário pode navegar pelo Samsung Ocean vendo fotos 360 do ambiente. Utilizamos uma câmera Samsung Gear 360 (Figura 24) para a captura das fotos, e o Unity 2017.1 para o desenvolvimento do projeto.

Figura 24 – Câmera Gear 360

11.1 Preparando o Unity para VR

Pode-se fazer o download do Unity no link https://unity3d.com/pt/get-unity/download.

O Unity precisa estar configurado para trabalhar com desenvolvimento android, no caso, e precisa do JDK disponível em http://www.oracle.com/technetwork/java/javase/downloads/index.html, android studio https://developer.android.com/studio/index.html e o Unity Android Support https://unity3d.com/pt/unity/whats-new/unity-2017.1.0.

O android SDK precisa dos seguintes componentes:

1. Android SDK Platform-Tools versão 23 ou superior, de preferência a mais atual.
2. Android Build Tools versão 19 ou superior, de preferência a mais atual.
3. API 19 ou superior, de preferência a mais atual.
4. Google USB driver.

Agora é possível criar um projeto Unity para trabalhar com VR mobile.

Para isso, é necessário mudar algumas configurações do projeto:

1. No Unity, configurar o JDK e o android SDK nas preferências.
2. Mudar o build settings para Android.
3. No player settings, selecionar Virtual Reality Supported.
4. Selecionar Oculus para GearVR ou Cardboard para google cardboard.
5. Selecionar Minimum API level para 19.

Com essas configurações é possível desenvolver e implementar aplicativos e experiências em VR no GearVR ou Cardboard.

11.2 Passeio virtual em VR

Para o local representado pela planta baixa da Figura 25 foi desenvolvido uma aplicação chamada de “passeio virtual”, uma aplicação simples que utiliza de diversas fotos de um local para criar uma experiência de navegação nesse espaço, utilizando VR.

Os pontos dessas fotos foram escolhidos de forma que o usuário consiga olhar o ponto de outra foto, sendo possível criar uma navegação entre esses pontos.

Figura 25 – Planta baixa do ocean indicando os pontos onde foram tiradas as fotos 360.

Alguns cuidados são necessários na hora de tirar essas fotos.

Estabilidade e altura da câmera: Utiliza-se um tripé para estabilizar a câmera e deixá-la numa altura agradável, mais ou menos a altura dos olhos de uma pessoa padrão (1,70 m). Se a foto estiver num ângulo torto em relação ao horizonte, quando colocado no VR o usuário pode sentir uma certa dissociação do movimento, a sua rotação de cabeça não vai corresponder a visualização dele do que ele acha ser o horizonte, causando desconforto.

Iluminação: dependendo da iluminação do ambiente algumas partes das fotos ficarão mais claras que outras, sendo possível que estas partes claras sejam difíceis de visualizar no VR. Ao tirar a foto deve-se procurar minimizar esses pontos fortes de luz. É possível ajustar a iluminação do ambiente com pós-produção.

Objetos próximos da câmera: Algumas pessoas relatam um certo desconforto quando há objetos próximos à câmera, como mesas e cadeiras, dando uma sensação de flutuar. Acredita-se que essa sensação é causada pela falta do corpo na experiência VR. Preferivelmente a câmera deve ser colocada em um ambiente amplo, sem objetos próximos.

Além das fotos, faz-se necessário alguns elementos de interface para trabalhar a interação do usuário com o ambiente virtual. No caso, utilizou-se apenas um botão, para fazer transição entre as cenas com as fotos do ambiente, uma mira que indica para onde o usuário está olhando e uma barra circular, que vai ficar junto com a mira, para indicar o tempo de carregamento antes de executar uma ação em um objeto (Figura 26).

Figura 26 - Botão, Barra circular e Mira

11.3 Adicionando fotos em 360 graus ao Unity

Depois de tiradas as fotos, pode-se importá-las para o Unity, porém deve-se mudar o *import settings* da textura no Unity, o padrão é que a texture esteja com a propriedade *Texture Shape* como 2D, e precisa-se que ela seja do tipo Cube, para poder usá-la como *cubemap*.

Assim, a imagem será interpretada como um *cubemap*, e pode ser utilizada para um ambiente imersivo 360.

Para colocar a imagem na cena Unity, precisa-se de uma esfera nessa cena, com um material próprio que vai usar um *shader* criado para renderizar a textura *cubemap* por dentro da esfera (Anexo 1).

Para visualizar corretamente essa foto 360, deve-se colocá-la centralizada com a MainCamera da cena.

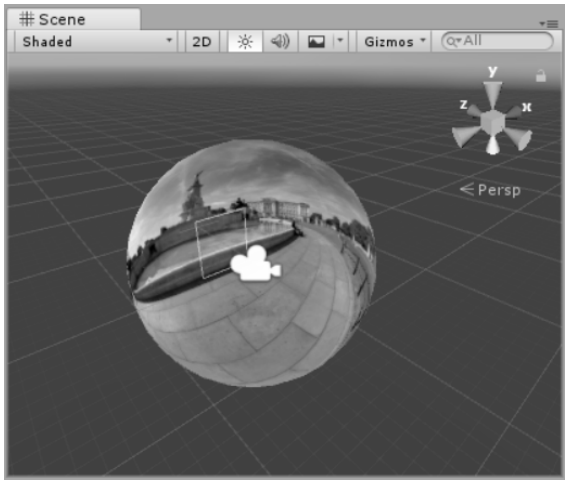

Figura 27 - Esfera com texture cubemap

Outro detalhe é a distância da foto em relação a câmera. Objetos muito próximos da câmera são difíceis de focar em VR. Aconselha-se mudar a escala da esfera para distanciá-la da câmera, colocando-a em (20,20,20).

Agora tem-se uma foto 360 visualizada dentro do Unity, pode-se colocar essa cena no dispositivo VR. Se tudo estiver configurado, é possível criar uma *build* no dispositivo e visualizá-lo.

Para tanto o dispositivo deve estar conetado pelo usb, no *build settings* do Unity selecione *build and run*, renomeie o apk. O Unity fará o processo de *build* e executará o aplicativo *android* no dispositivo conectado.

11.4 Interação em VR no Unity

Como dito anteriormente a melhor forma de fazer interação em VR no mobile é com *Input Gaze*, pode-se então implementar um sistema desse tipo no Unity com facilidade.

Para o projeto em discussão, a forma de interação é realizada com botões que estão posicionados na cena, quando o usuário olha para esse botão, começa a carregar uma barra, depois que a barra carrega, a outra cena correspondente ao botão é carregada, com um efeito de fade in/fade out para evitar possíveis problemas se o dispositivo demorar no carregamento de uma nova cena, perdendo o *tracking* da cabeça e causando desconforto no usuário [6].

Para isso foi criado um sistema de *input gaze* simples, usando *raycast* a partir da câmera, e com eventos que são chamados a partir desse *raycast*.

Raycast é o processo de traçar um raio, de um ponto, em uma direção, e verificar os objetos com a qual esse raio colidiu. No caso, fez-se um *raycast* a partir da posição da câmera em direção a frente da câmera.

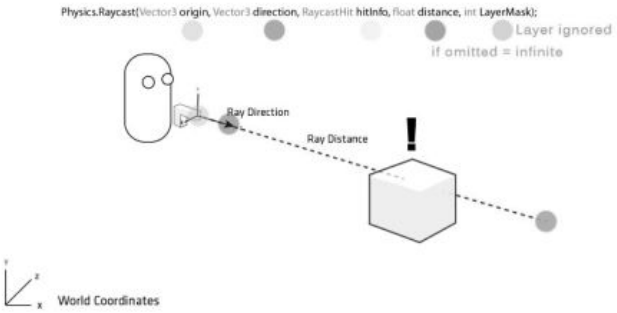

Figura 28 - Raycast

Precisa-se de alguns objetos na MainCamera: uma mira e uma barra circular que vai ser carregada quando o usuário ficar olhando para um objeto. É possível criar esses objetos usando os elementos de UI do Unity. No caso, será necessário:

1. Criar um Canvas: GameObject -> UI-> Canvas.
2. Mudar o nome do objeto para CameraCanvas
3. No componente Canvas, mudar o Render Mode para Screen Space - Camera.
4. Na propriedade Render Camera, colocar a MainCamera da cena.
5. Mudar o plane Distance para 4.
6. Mudar o Render Mode para World Space.
7. Remover o componente Graphics Raycast.
8. Colocar o canvas como filho da MainCamera na hierarchy.
9. Adicionar um Image no canvas.
10. GameObject → UI → Image.
11. Mudar o nome do objeto para Mira.
12. Centralizar a imagem.
13. No componente Image, definir o sprite para o sprite da Mira.
14. No componente RectTransform, definir o width e height como 20 em ambos.
15. Criar um objeto UI → Image.
16. Selecionar o sprite da barra circular para esse objeto (Figura 28).
17. Mudar o width e height para 40.
18. No componente Image, mudar o Image Type para filled.
19. Fill Method: radial 360.
20. Fill Origin: Top.
21. Fill Amount: 0.

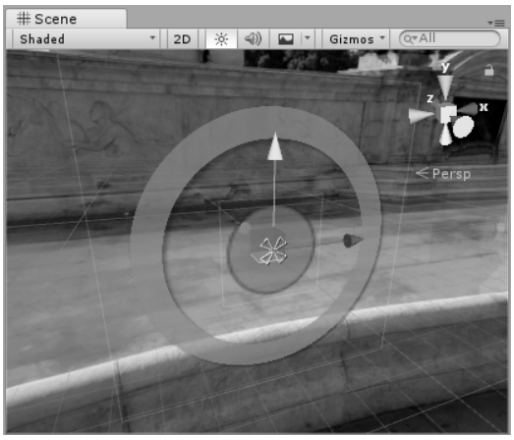

Figura 29 – Mira e barra circular

Na MainCamera adiciona-se um *script* que vai fazer um *raycast* a partir da câmera na direção que o usuário está olhando (Anexo 2).

Quando esse *raycast* encontra um objeto, ele vai verificar a existência de um *script* RaycastTarget.cs, e vai chamar os eventos correspondentes nesse *script* (Anexo 3).

Além disso, também deve ser chamado o evento para iniciar a barra de carregamento circular, e um evento quando essa barra terminar de carregar (Anexo 4).

Esses 3 *scripts* combinados são a base desse sistema de *input gaze*, agora é possível chamar determinar eventos quando o usuário olhar para um objeto, deixar de olhar para um objeto e terminar de carregar a barra circular.

Para fazer o fade entre cenas, é preciso um *panel* que fique na frente da câmera e cubra toda a tela. É possível criar um usando os elementos de UI dos Unity.

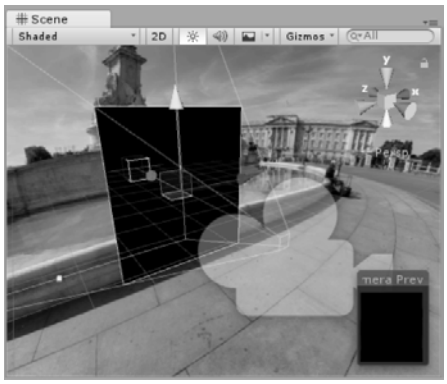

Figura 30 – *Panel* escuro na câmera

Utiliza-se dois *scripts*, um para fazer o fade in/fade out do *panel* (deixando-o transparente ou opaco) (Anexo 5) e outro para carregar a outra cena assim que o fade é terminado (Anexo 6).

Agora que se tem os componentes necessários para interagir com objetos, e a mudança de cena com fade, pode-se utilizar os elementos de interface do Unity para criar um botão, para quando o usuário olhar para esse botão, é chamado um evento de mudança de cena.

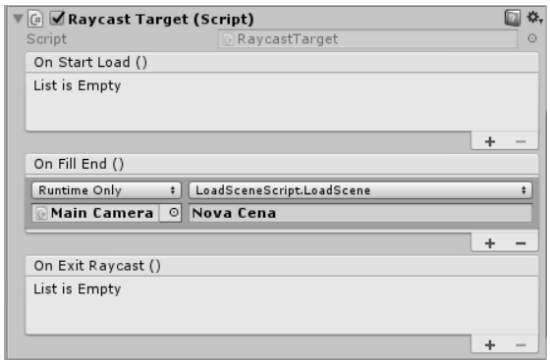

Figura 31 - Componente RaycastTarget com o LoadScene da câmera chamado no evento OnFillEnd

Pode-se utilizar o RaycastTarget para trabalhar com diversas ações como chamada de funções em script, animações e ativar/desativar componentes.

12 POSSIBILIDADES

Com esses *scripts* é possível criar uma experiência de passeio virtual, utilizando botões na cena que mudam o ponto de vista do usuário e o ambiente em que eles se encontram. Ou então um visualizador de fotos, com várias fotos 360 dispostas num menu espacial.

Jogos podem se beneficiar desse modelo de desenvolvimento, ambientes pré-renderizados que já são usados em jogos desde o início do desenvolvimento em 3D, e jogos com captura real de imagem e vídeo, conhecidos como jogos em FMV, sempre tiveram uma importância no mercado de jogos e são ainda uma possibilidade de desenvolvimento de experiências interessantes [13].

O mercado de arquitetura e eventos já vem mostrando um grande interesse em VR, esse projeto exemplifica um exemplo de aplicação que é muito usada para demonstrar ambientes reais, com fotos 360, para pessoas interessadas, ou então visualização de vídeos em 360.

O mercado de turismo se beneficiaria de uma aplicação para mostrar pontos turísticos e locais de interesse em VR [10].

13 CONSIDERAÇÕES FINAIS

O mercado de VR ainda possui alguns desafios para se firmar na realidade das pessoas. Apesar de muitas previsões apontarem a realidade virtual como uma nova forma de interação homem-computador, as plataformas atuais ainda possuem dificuldade em penetrar no dia a dia do público geral, seja por questões técnicas de hardware, quanto por questões comportamentais. Acredita-se que a VR esteja num impasse em relação ao mercado, empresas tem receio de investir no mercado com uma pequena base de usuários instalada, e usuário tem receio de adquirir um produto com pouco suporte das empresas. Para ultrapassar isso, os desenvolvedores de VR podem utilizar as plataformas existentes para criar experiências interessantes, e ferramentas inovadoras que possam dar uma pequena prévia do que o futuro da tecnologia VR pode trazer. Dessa forma, os usuários mantêm interesse nas possibilidades que essas plataformas podem trazer, e o mercado continua investindo em experiencias e ferramentas para melhorar essas experiências [6].

Apesar das limitações atuais, é possível criar aplicativos interessantes em VR. As ferramentas estão disponíveis, e a expectativa do publico e do mercado está em busca de alguma experiência que alavanque a realidade virtual para o status de

"realidade", algo que fará parte das nossas vidas como o computador pessoal e o *smartphone*.

14 AGRADECIMENTOS

Os resultados apresentados nesta publicação foram obtidos por meio de atividades de Pesquisa e Desenvolvimento do projeto SAMSUNG OCEAN, uma parceria entre a Universidade do Estado do Amazonas e a Samsung Eletrônica da Amazônia Ltda., apoiado pela SUFRAMA sob os termos da lei federal nº 8.248/91.

REFERÊNCIAS

[1] T. Parisi, *Learning Virtual Reality*: Developing Immersive Experiences and Applications for Desktop, Web, and Mobile. O'Reilly Media, Sebastopol (2015).

[2] S. Mandal, Brief Introduction of Virtual Reality & its Challenges. *International Journal of Scientific & Engineering Research*, Volume 4, Issue 4, April, 2013.

[3] A. S. Alqahtani, L. F. Daghestani, and L. F. Ibrahim, Environments and System Types of Virtual Reality Technology in STEM: a Survey. *International Journal of Advanced Computer Science and Applications (IJACSA)*. 8(6), 2017

[4] L. D. Croche, *et al*. Realidade virtual – A viabilidade da imersão total na atualidade. *Revista Contribuciones a las Ciencias Sociales*, (2016).

[5] H. Bellini, W. Chen, M. Sugiyama, M. Shin, S. Alam, and D. Takayama. Virtual and Augmented Reality: Understanding the race for the next computing platform. Disponível em: <http://www.goldmansachs.com/our-thinking/pages/technology - driving-innovation-folder/virtual-and-augmented-reality/report.pdf>. Acesso em: 24 de setembro de 2017.

[6] R. Gleasure, J. Feller. A Rift in the Ground: Theorizing the Evolution of Anchor Values in Crowdfunding Communities through the Oculus Rift Case Study. *Journal of the Association for Information Systems*. Vol. 17: Iss. 10, Article 1. (2016).

[7] SuperData Research | Games data and market research » Unity and SuperData launch major mobile games and VR report. Disponível em: https://www.superdataresearch.com/unity-and-superdata-launch-major-mobile-games-and-vr-report/. Acesso em: 24 de setembro de 2017.

[8] A new dimension - Designing for Google Cardboard. Disponível em: https://designguidelines.withgoogle.com/cardboard/designing-for-google-cardboard/a-new-dimension.html. Acesso em: 24 de setembro de 2017.

[9] Mobile Best Practices. Disponível em: <https://developer.oculus.com/documentation/mobilesdk/latest/concepts/book-mobile-best-practices/>. Acesso em: 24 de setembro de 2017.

[10] Experiences | Oculus. (2017). Experiences | Oculus. Disponível em: https://www.oculus.com/experiences/gear-vr/. Acesso em: 24 de setembro de 2017.

[11] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, v.6 n.11, p.21-29, November 1986.

[12] D. Bowman, *et al*. An introduction to 3-D user interface design. Presence: Teleoperators and virtual environments. 10.1: 96-108. (2001).

[13] D. Arsenault and B. Perron. De-framing video games from the light of cinema. *G|A|M|E – Games as Art, Media, Entertainment* 1.4 (2015).

[14] R. Scott, *Level Up!* The guide to great video game design. John Wiley & Sons, 2014.

[15] S. Manjrekar, S. Sandilya, D. Bhosale, S. Kanchi, A. Pitkar, M. Gondhalekar, CAVE: An Emerging Immersive Technology-A Review, *Computer Modelling and Simulation (UKSim)*, pp. 131-136, 2014.

ANEXOS

Anexo 1 – Shader Cubemap.shader

```
Shader "Unlit/CubeMapShader"
{
  //Cubemap usado no shader
  Properties {
    _Cube("cube map", Cube) = "" {}
  }
  SubShader {
    Pass {
      //Renderiza o objeto por dentro
      cull front
      CGPROGRAM
      #pragma vertex vert
      #pragma fragment frag

      #include "UnityCG.cginc"

      uniform samplerCUBE _Cube;

      struct appdata {
        float4 vertex: POSITION;
      };

      struct v2f {
        float3 uv: TEXCOORD0;
        float4 vertex: SV_POSITION;
      };

      v2f vert (appdata v) {
        v2f o;
        o.uv = v.vertex;
        o.vertex =
UnityObjectToClipPos(v.vertex);
        return o;
      }
      //Renderiza o cubemap como
      //textura do material
      fixed4 frag (v2f i): COLOR {
        float4 result = texCUBE(_Cube, i.uv);
        return result;
      }
      ENDCG
    }
  }
}
```

Anexo 2 – Script Raycast.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Raycast : MonoBehaviour {
  //Objeto mira, Tamanho inicial da mira
  //e distância maxima da mira em relação a camera
  public Transform reticle;
  private Vector3 startScale;
  public float MaxDistanceReticle;

  //Objeto alvo do raycast e
  //O componente RaycastTarget desse objeto
  private GameObject ActualTarget;
  public RaycastTarget ActualTargetEvents;

  //Compoente de controle da barra circular
  LoadCircle CircleScript;
  // Use this for initialization
  void Start () {
    //Inicializa a variavel StartScale usada
    //para calcular o tamanho da mira em relação
    //a distância ao objeto que está na mira.
    float dist =
       Vector3.Distance(transform.position,
       reticle.position);
    startScale = reticle.localScale / dist;

    //Pega o Componente LoadCircle desse objeto.
    CircleScript = GetComponent<LoadCircle>();
  }


  void Update () {
    //Raycast a partir desse objeto camera,
    //na direção que o usuário está olhando
    Ray ray = new Ray(transform.position,
       transform.forward);
    RaycastHit hit;
    Physics.Raycast(ray, out hit);
    //Verifica o objeto que raycast colide
    if(hit.collider != null){
      //Verifica se o raycast mudou o objeto alvo
      if(ActualTarget != hit.collider.gameObject){
        ActualTarget = hit.collider.gameObject;
        //Verifica se tem um componente
        //RaycastTarget nesse objeto
        if(ActualTarget.GetComponent
         <RaycastTarget>() != null){
          ActualTargetEvents =
          ActualTarget.GetComponent
          <RaycastTarget>();
        }else{
          ActualTargetEvents = null;
        }
        //Inicia o evento de carregamento
        //da barra circular
        CircleScript.StartFill
        (ActualTargetEvents);
      }
    //Ajusta a posição, tamanho e rotação da mira
    // a partir do ponto de colisão do raycast
    reticle.position = hit.point;
    reticle.localScale = startScale *
                         hit.distance;
    reticle.rotation = Quaternion.FromToRotation
                   (Vector3.forward, hit.normal);
    //Se não estiver olhando para nenhum objeto
    } else {
      ActualTarget = null;
      //Chama o evento de parar o
      //carregamento da barra circular
      CircleScript.ResetFill(ActualTargetEvents);
      //Ajusta a posição, tamanho e
      //rotação da mira
      // em relação à distancia máxima
      reticle.position = ray.GetPoint
                         (MaxDistanceReticle);
      reticle.localScale = startScale *
                           MaxDistanceReticle;
      reticle.rotation = Quaternion.FromToRotation
                 (Vector3.forward, ray.direction);
    }
  }
}
```

Anexo 3 – Script RaycastTarget.cs

```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.EventSystems;

public class RaycastTarget : MonoBehaviour {
  //Eventos quando o usuario olha para esse
objeto,
  //quando ele deixa de olhar para um objeto e
  //quando termina de carregar a barra circular
  public UnityEvent OnStartLoad;
  public UnityEvent OnExitRaycast;
  public UnityEvent OnFillEnd;

  //Simula os eventos do Untiy com o evento da
mira
  void Start () {
    OnStartLoad.AddListener(() =>
SimulateOnEnter());
    OnFillEnd.AddListener(() => SimulateClick());
    OnExitRaycast.AddListener(() =>
SimulateOnExit());
  }

  void SimulateOnEnter(){
    ExecuteEvents.Execute(gameObject,
    new PointerEventData(EventSystem.current),
    ExecuteEvents.pointerEnterHandler);
  }

  void SimulateClick(){
    ExecuteEvents.Execute(gameObject,
    new PointerEventData(EventSystem.current),
    ExecuteEvents.pointerDownHandler);
    ExecuteEvents.Execute(gameObject,
    new PointerEventData(EventSystem.current),
    ExecuteEvents.pointerClickHandler);
  }

  void SimulateOnExit(){
    ExecuteEvents.Execute(gameObject,
    new PointerEventData(EventSystem.current),
    ExecuteEvents.pointerUpHandler);
    ExecuteEvents.Execute(gameObject,
    new PointerEventData(EventSystem.current),
    ExecuteEvents.pointerExitHandler);
  }
}
```

Anexo 4 – Script LoadCircle.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Events;

public class LoadCircle : MonoBehaviour {
  //Eventos da barra circular
  //quando o usuário olha para um objeto
  //quando o usuário deixa de olhar para um objeto
  //quando termina de carregar a barra circular
  public UnityEvent OnExitRaycast;
  public UnityEvent OnStartLoad;
  public UnityEvent OnFillEnd;

  //Barra circular
  public Image RadialBar;

  //Porcentagem de carregamento da barra
  private float loadFillAmount;
  //Tempo de carregamento total da barra
  public float LoadFillTime;
  //Co-rotina de carregamento da barra
  Coroutine FillCoroutine;
  IEnumerator Filling(RaycastTarget target){
    //Carregar a barra circular
    // no tempo determinado
    loadFillAmount = 0;
    while(loadFillAmount < 1){
      loadFillAmount += Time.deltaTime /
                        LoadFillTime;
      RadialBar.fillAmount = loadFillAmount;
      yield return new WaitForEndOfFrame();
    }
    loadFillAmount = 1;
    //Chama os eventos quando a barra
    //terminar de carregar
    OnFillEnd.Invoke();
    if(target != null)
      target.OnFillEnd.Invoke();
  }

  public void StartFill(RaycastTarget target){
    //Chama os eventos quando a
    //barra começa a carregar
    if(target != null)
      target.OnStartLoad.Invoke();
      OnStartLoad.Invoke();
      //Chama a co-rotina para iniciar o
      //carregamento da barra
      FillCoroutine = StartCoroutine
                      (Filling(target));
  }

  public void ResetFill(RaycastTarget target){
    //Chama os eventos quando
    //deixa de carregar a barra
    if(target != null)
      target.OnExitRaycast.Invoke();
      OnExitRaycast.Invoke();
      //Reinicia a barra de carregamento
      //Para a co-rotina de carregamento da barra
      RadialBar.fillAmount = 0;
      if(FillCoroutine != null)
        StopCoroutine(FillCoroutine);
  }
}
```

Anexo 5 – Script FadeScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class FadeScript : MonoBehaviour {
  Image panel;
  //Inicializa o panel como escuro
  //e chama a co-rotina FadeOut
  void Start () {
    panel = GetComponent<Image>();
    panel.color = Color.black;
    StartCoroutine(Fade(Color.black,
Color.clear));
  }

  //Função para iniciar o FadeIn
  public void StartFade(){
    StartCoroutine(Fade(Color.clear,
Color.black));
  }
  //Co-rotina de Fade
```

```
  public IEnumerator Fade(Color start, Color end){
    float value = 0;
    while(value < 1){
      yield return new WaitForEndOfFrame();
      value += Time.deltaTime;
      panel.color = Color.Lerp(start, end, value);
    }
    panel.color = end;
  }
}
```

Anexo 6 – Script LoadSceneScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LoadSceneScript : MonoBehaviour {
  //Script que controla o FadePanel
  public FadeScript fade;
  //Função para iniciar o
  //carregamento da cena
  public void LoadScene(string scene){
    StartCoroutine(FadeAndLoad(scene));
  }
  //Espera o fade para carregar a nova cena
  IEnumerator FadeAndLoad(string scene){
    yield return StartCoroutine
    (fade.FadeInCoroutine());

    SceneManager.LoadScene(scene);
  }
}
```