

DESENVOLVIMENTO DE BIBLIOTECA PARA PROJEÇÃO EM PIRÂMIDE HOLOGRÁFICA

Lucas Matheus Westphal, Dalton Solano dos Reis – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

lmwestphal@furb.br, dalton@furb.br

Resumo: O presente artigo apresenta o desenvolvimento de uma biblioteca para auxílio na construção de cenas utilizando uma pirâmide holográfica. A implementação da biblioteca se deu com duas estratégias distintas, uma duplicando o alvo da projeção a outra capturando o retorno de câmeras presentes na cena em imagens posicionadas na tela para construir os pontos de visão necessários. As duas estratégias foram comparadas constatando-se que a segunda estratégia economiza recursos de processamento e memória ao renderizar a cena. Foi desenvolvida também uma aplicação de demonstração utilizando a biblioteca implementada com o intuito de comparar a projeção na pirâmide com outros modos de visualização de modelos 3D, sendo eles o modo normal e modo utilizando o CardBoard (realidade virtual).

Palavras-chave: Pirâmide holográfica. Fantasma de Pepper. Projeção. Unity

1 INTRODUÇÃO

Atualmente a forma de representação de informação no cotidiano ocorre por meio de áudio e vídeo. Para Hoffman (2018), é muito importante para a sociedade inovar o método de representação das aplicações do cotidiano, pois o mercado audiovisual está em constante desenvolvimento. Para tal feito, algumas técnicas de projeção holográfica são utilizadas, sendo as mais conhecidas o Head-Up Display (HUD), o fantasma de Pepper e a pirâmide holográfica. Segundo Schivani *et al.* (2018), as imagens observadas através das pirâmides, classificadas de maneira errônea como holográficas, utilizam a técnica do Fantasma de Pepper. São imagens bidimensionais (2D) e não hologramas como são definidas popularmente, gerando assim o erro conceitual.

A holografia (do grego *holos*: todo, inteiro e *graphos*: sinal, escrita) é segundo Gabor (1971), um meio de registro “integral” da informação, com relevo e profundidade, que se obtém a partir da divisão das ondas luminosas e seu padrão de interferência sobre um objeto. Por esse motivo, um holograma possui propriedades diferentes dos meios tradicionais de visualização da informação e tem segundo Ferreira e Lopes (2017), várias aplicações no mundo, tanto científicas quanto artísticas ou em questões de segurança. Segundo Rebordão (1989), os hologramas são popularmente conhecidos como fotografias 3D. Os filmes de ficção científica, as exposições em que têm circulado, vulgarizaram o holograma embora pouco tenham contribuído para elucidar a sua estrutura e construção (REBORDÃO, 1989).

Tendo em vista o cenário de inovações da representação, as várias *engines* gráficas atuais como a Unity possuem inúmeras rotinas gráficas para auxiliar a representar a informação, desde a representação 2D, até a estereoscopia, porém não existe uma rotina que auxilie o desenvolvimento de cenas para a projeção 3D (UNITY, 2019). Uma forma de disponibilizar este auxílio é por meio de bibliotecas, que podem ser em formato de *asset* ou *packages*. Com essa possibilidade, este trabalho desenvolveu uma biblioteca que permite gerar cenas 3D para a projeção na pirâmide holográfica, bem como disponibilizar uma aplicação de testes que inclui outras formas de visualização do modelo com o intuito de compará-las para testes. Também disponibiliza uma rotina que permite calcular as dimensões para auxiliar a montagem da pirâmide holográfica.

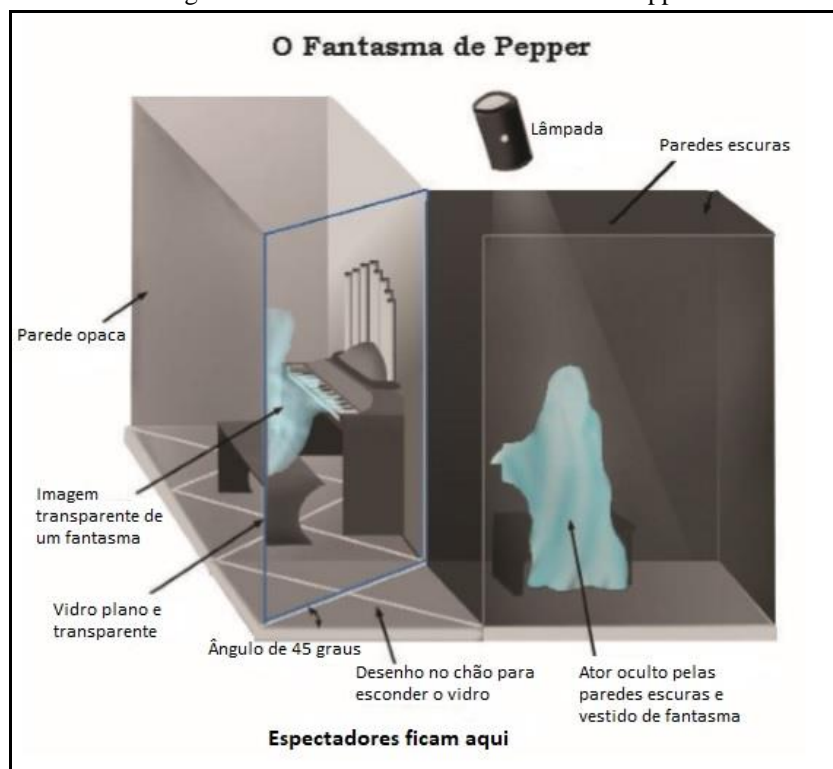
2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentados conceitos e ferramentas utilizados no desenvolvimento da biblioteca. Serão abordados os assuntos Fantasma de Pepper, técnica utilizada para simular o efeito de uma figura fantasmagórica em teatros e apresentações, e a pirâmide holográfica que utiliza os conceitos da técnica de Pepper para seu funcionamento. Por fim, são apresentados os trabalhos correlatos a este, que utilizam a pirâmide holográfica como modo de visualização.

2.1 FANTASMA DE PEPPER

O Fantasma de Pepper ou “Casa de Monga”, como é conhecido no Brasil é segundo Medeiros (2006), uma das ilusões de ótica mais fascinantes e conhecidas em todo o mundo. Trata-se da reflexão de um objeto, oculto dos observadores, em uma lâmina de vidro plana inclinada a 45°, desenvolvida pelo professor de Química inglês John Henry Pepper. A Figura 1 demonstra o funcionamento da técnica de Pepper, utilizando como exemplo uma figura fantasmagórica tocando um piano.

Figura 1 - Funcionamento do Fantasma de Pepper



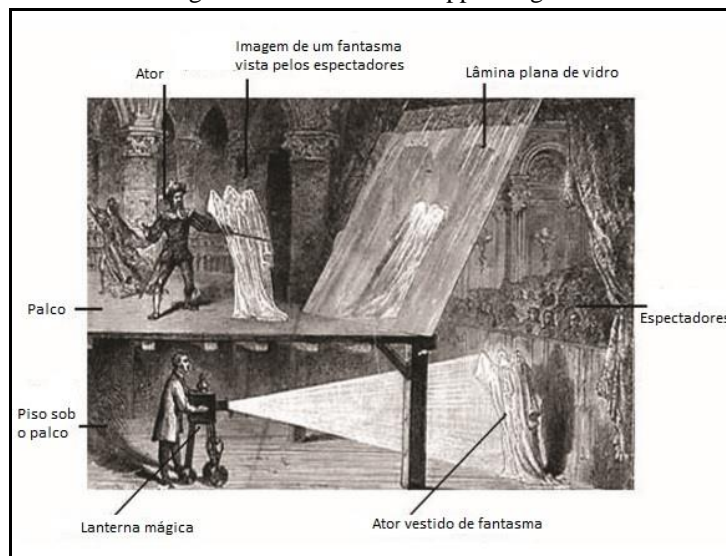
Fonte: Medeiros (2006), alterado pelo autor.

Percebe-se, na Figura 1, que conforme explica Medeiros (2006), existem dois compartimentos de mesmo tamanho, perpendiculares entre si. No primeiro, com as paredes opacas e visível ao público, se encontra apenas um piano, enquanto no compartimento vizinho, com paredes escuras e oculto do público, se encontra um ator vestido de fantasma, encenando tocar o piano. Segundo o autor, a posição do banco na frente do piano e a do ator são equidistantes em relação à lâmina de vidro inclinada a 45° que é mostrada na figura. O autor também aponta que existe algum tipo de obstáculo que não permite que o público se aproxime da cena o suficiente para ver o interior do compartimento onde se encontra o ator.

Inicialmente, segundo Medeiros (2006), o compartimento com o piano é intensamente iluminado, enquanto o outro é deixado às escuras. À proporção que a luz do compartimento com o piano é reduzida, se aumenta a iluminação do compartimento com o ator, fazendo assim com que sua imagem seja gradativamente refletida no vidro inclinado na direção do público, cada vez com mais intensidade. O autor constata também que conforme a iluminação sobre o ator aumenta, mais nítido se torna o reflexo, perdendo deste modo o seu aspecto fantasmagórico. Assim, segundo o autor, é possível produzir imagens com aspectos mais ou menos fantasmagóricos através do controle da iluminação dos ambientes.

Na versão original levada aos palcos no século XIX pelo professor Pepper, segundo Medeiros (2006), a imagem de um fantasma era produzida com o auxílio de uma lanterna mágica, um antigo projetor, posicionada no piso inferior do palco, conforme demonstra a Figura 2. O autor afirma que, de acordo com o ângulo de visão dos espectadores, a ilusão produzida pela encenação de Pepper era perfeita sendo uma das muitas ilusões de ótica que fez sucesso no século XIX e que ainda surpreende quem a observa sem ter conhecimento em ótica.

Figura 2 - Fantasma de Pepper original



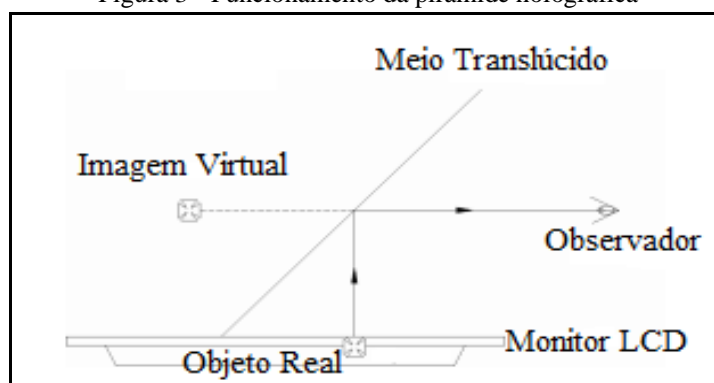
Fonte: Medeiros (2006), alterado pelo autor.

Medeiros (2006) conclui que existe no uso pedagógico da técnica, uma possibilidade inegável de cativar a atenção dos estudantes para a aprendizagem, além da curiosidade histórica dos artefatos mencionados em seu trabalho.

2.2 PIRÂMIDE HOLOGRÁFICA

A pirâmide holográfica é um utensílio montado geralmente de acrílico, que é utilizado para projetar objetos 3D. Segundo Schivani *et al.* (2018), as imagens observadas através das pirâmides utilizam a técnica do “Fantasma de Pepper”. A Figura 3 demonstra o funcionamento da pirâmide com a imagem original exibida por um display que pode ser um smartphone, TV, monitor ou tablet, por exemplo, e que por sua vez é refletida pelas faces da pirâmide segundo conceitos da técnica de Pepper.

Figura 3 - Funcionamento da pirâmide holográfica



Fonte: Schivani *et al.* (2018).

Para o funcionamento correto da pirâmide, Schiviani *et al.* (2018) destacam que a face central do display deve ser posicionada no topo da pirâmide e o conjunto pirâmide-display pode ser disposto com a base da pirâmide voltada tanto para baixo como para cima. Cada segmento da pirâmide reflete uma imagem diferente ou o mesmo objeto com pontos de vista diferentes, simulando um objeto tridimensional.

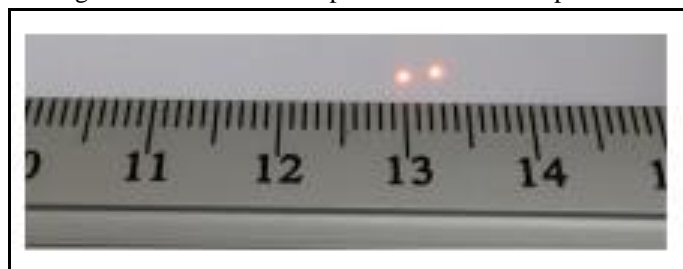
Conforme constatado por Schivani *et al.* (2018), quando a pirâmide foi feita em proporção para um display pequeno, como um smartphone, não havia problema na projeção da imagem, porém quando as proporções aumentaram, para um monitor de 14 polegadas, ocorreu um efeito de duplicata da imagem, como pode ser visto na Figura 4. Os autores, após realizarem cálculos sobre a refração e reflexão dos raios incidentes sobre as faces da pirâmide chegaram à conclusão de que este efeito é decorrente de um desvio do raio incidente ao sofrer reflexão interna e refração, passando do ar para a pirâmide e de volta para o ar. Devido à espessura do material utilizado, no caso acrílico de 4mm, o desvio recorrente deste processo se torna mais acentuado e perceptível a olho nu. A Figura 5 demonstra o efeito de desvio sofrido por um raio laser no mesmo ambiente que foi constatado o desvio na projeção, respeitando a iluminação do ambiente e do monitor sobre a pirâmide.

Figura 4 – Efeito de duplicata na pirâmide



Fonte: Schivani *et al.* (2018).

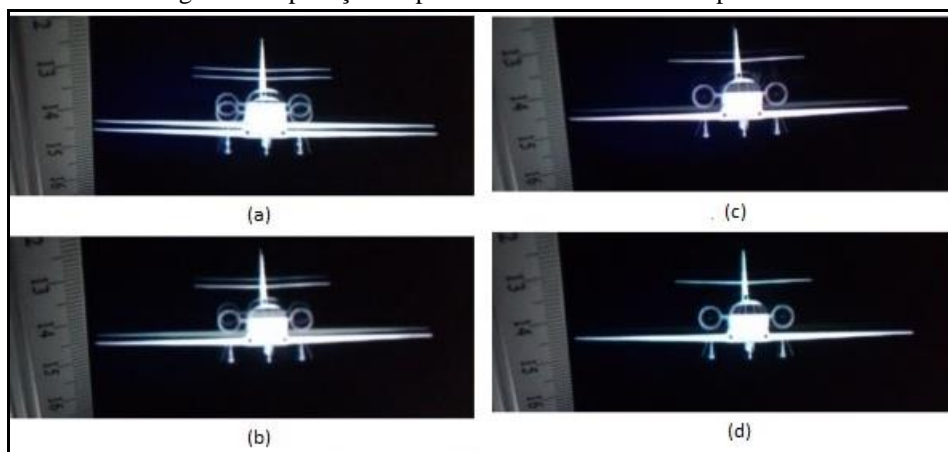
Figura 5 - Desvio sofrido pelos raios de luz na pirâmide



Fonte: Schivani *et al.* (2018).

Para solucionar o problema de duplicata na pirâmide, foram utilizadas por Schivani *et al.* (2018) películas reflexivas, como as presentes em janelas de carros e portas de vidro, por terem a capacidade de diminuir a iluminância das imagens da face oposta da pirâmide, reduzindo o efeito de duplicata. Os autores utilizaram uma película automotiva genérica encontrada em comércio e classificada como #2. Após a aplicação dessas películas, notou-se que quanto mais películas eram sobrepostas, mais nítida a imagem parecia, e menos efeito de duplicata era visível. Conforme demonstra a Figura 6, a imagem A representa a pirâmide sem aplicação de película reflexiva, a imagem B representa a pirâmide com aplicação de uma camada de película, a imagem C com duas camadas e a imagem D com três camadas.

Figura 6 - Aplicação de película automotiva sobre a pirâmide



Fonte: Schivani *et al.* (2018), alterado pelo autor.

2.3 TRABALHOS CORRELATOS

Foram selecionados um produto e dois trabalhos que utilizam da mesma forma de representação gráfica ou *display*. No Quadro 1 é apresentado um produto que simula a captura de criaturas digitais (PIXELSAV, 2015). O primeiro trabalho se trata de um estudo sobre como a holografia pode ser utilizada ao se estudar fenômenos astronômicos (HOFFMAN, 2018), apresentado no Quadro 2. No Quadro 3 é descrito um trabalho sobre a aplicação de holografia utilizada para gerar um avatar da assistente virtual da Microsoft, a Cortana (ARCHER, 2017).

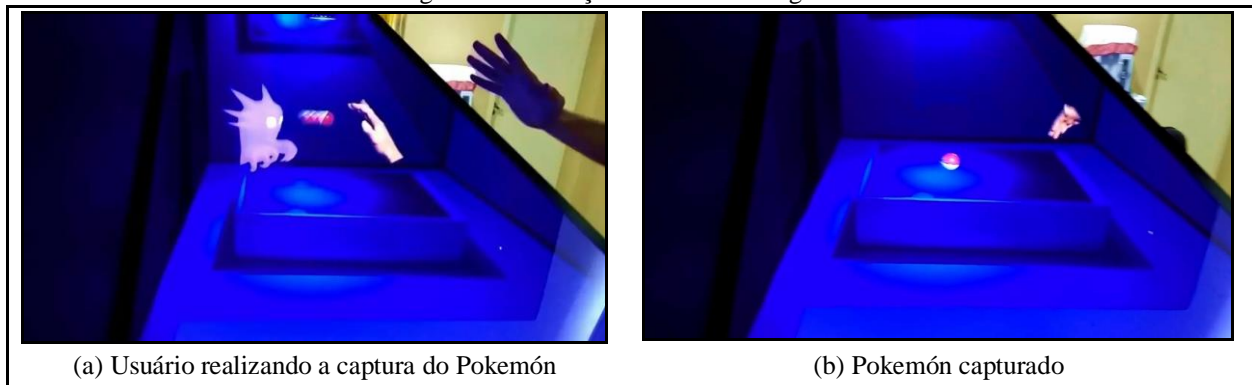
Quadro 1 – Totem holográfico: Pokemon

Referência	Pixelsav (2015)
Objetivos	Simular a captura de criaturas digitais.
Principais funcionalidades	Permitir que o usuário possa utilizar as próprias mãos para interagir com o produto.
Ferramentas de desenvolvimento	Por ser um produto comercial, não foram encontradas especificações técnicas.
Resultados e conclusões	Por ser um produto comercial, não foram encontrados dados de estudo sobre o assunto.

Fonte: elaborado pelo autor.

O Totem holográfico (PIXELSAV, 2015) é um display volumétrico que se utiliza da pirâmide holográfica, dando a impressão de que o objeto, ou criatura, está dentro do recipiente. As criaturas, no caso, são monstros virtuais da animação/jogo Pokémon, pertencente a empresa Nintendo. O usuário interage com o produto, utilizando as próprias mãos para alterar a cena projetada. A Figura 7-a demonstra o produto em utilização por um usuário que realiza a captura do Pokemon tipo fantasma Haunter. A captura do Pokemon se dá arremessando no mesmo uma esfera, chamada Pokebola, que prende a criatura em seu interior (Figura 7-b). Por se tratar de um produto comercial, não foram encontrados dados sobre seu desenvolvimento, ou testes e resultados de estudos.

Figura 7 - Utilização do Totem Holográfico



Fonte: Pixelsav (2015).

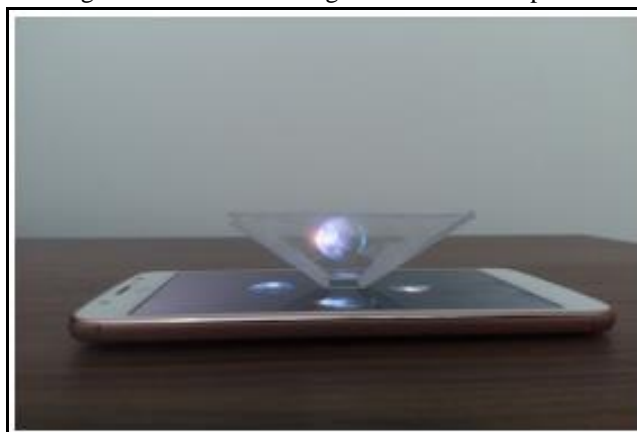
Quadro 2 – Estudo sobre a holografia aplicada à visualização do eclipse solar e lunar

Referência	Hoffman (2017)
Objetivos	Facilitar a visualização dos fenômenos astronômicos eclipse lunar e solar.
Principais funcionalidades	Permitir ao usuário visualizar como ocorre um eclipse em quatro pontos de vista, facilitando o entendimento da posição dos astros durante o fenômeno.
Ferramentas de desenvolvimento	Animação do fenômeno astronômico gerada pelo autor utilizando 3DS Max 2017, software disponibilizado pela Autodesk.
Resultados e conclusões	Os resultados do estudo foram satisfatórios, pois permitiram a visualização em escala reduzida de como ocorre um eclipse, e proporcionou a realização de um projetor holográfico a partir de uma pirâmide construída a partir de utensílios básicos.

Fonte: elaborado pelo autor.

Hoffman (2018) desenvolveu uma aplicação de projeção holográfica para facilitar a visualização do fenômeno astronômico eclipse, tanto lunar quanto solar. Para atingir tal objetivo, Hoffman (2018) utilizou a pirâmide holográfica, e a animação do fenômeno foi gerada pelo próprio autor. A Figura 8 demonstra um protótipo da pirâmide, utilizado como uma primeira experiência realizada por Hoffman (2018) com o dispositivo. Para esta experiência, foi utilizado um arquivo de vídeo pronto para a pirâmide, disponível na internet, e a estrutura foi montada em acetato, fixa com fita adesiva para fácil desmontagem caso necessitasse de ajustes estruturais.

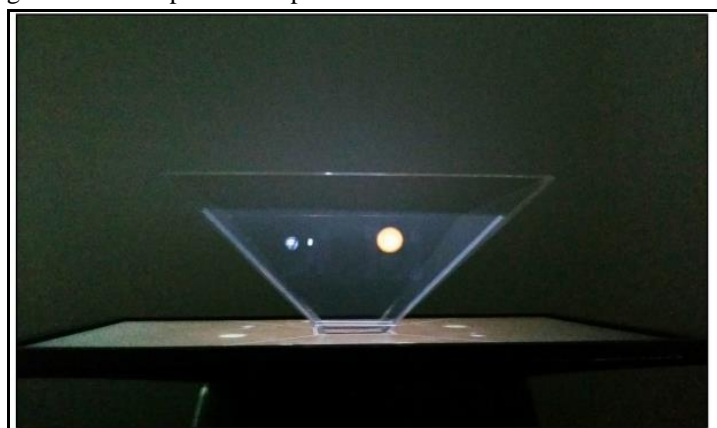
Figura 8 – Pirâmide holográfica sobre smartphone



Fonte: Hoffman (2018).

O resultado desta primeira experiência realizada por Hoffman (2018) foi satisfatório tendo em vista que proporcionou a realização de um projetor holográfico a partir de uma pirâmide construída com utensílios básicos. A animação do fenômeno astronômico foi gerada pelo autor, utilizando o software de modelagem e animação 3DS Max 2017, disponibilizado pela Autodesk. Um protótipo final da pirâmide foi construído pelo autor utilizando acrílico transparente com espessura de 4mm. A Figura 9 demonstra o protótipo final em utilização.

Figura 9 - Protótipo final da pirâmide de acrílico sobre monitor de LED



Fonte: Hoffman (2018).

Com a utilização do protótipo final, o autor constata o potencial didático proporcionado pelas pirâmides holográficas, gerando uma experiência de aprendizado diferenciada, podendo fomentar maior interesse do público escolar.

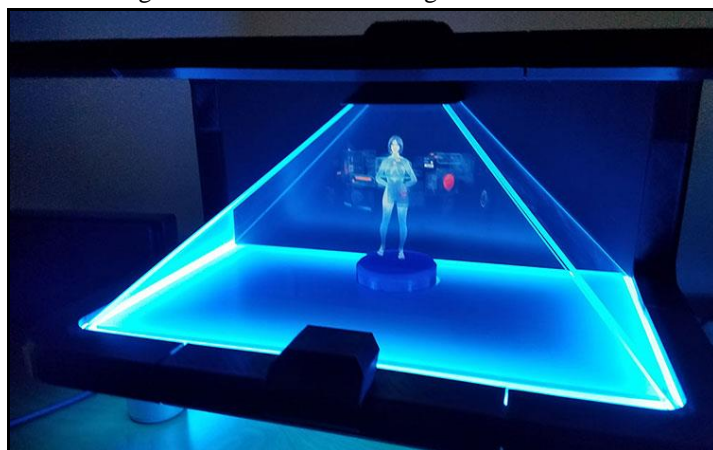
Quadro 3 – Holographic Cortana appliance: Working concept

Referência	Archer (2017)
Objetivos	Demonstrar um avatar para a assistente virtual Cortana.
Principais funcionalidades	Permitir que o usuário possa utilizar comandos de voz para interagir com a aplicação. Permitir que o usuário visualize o avatar a partir da pirâmide holográfica.
Ferramentas de desenvolvimento	Aplicação para gerar o avatar feita em Unity. Um serviço <i>proxy</i> que verifica os retornos da Cortana. Iluminação controlada por um microcontrolador Arduino. Aplicação de reconhecimento facial.
Resultados e conclusões	Os resultados foram satisfatórios, apesar dos problemas encontrados. Foi possível fazer pesquisas de base de conhecimento com a Cortana além poder ser estendido para automação residencial e tarefas de música, por utilizar a Cortana nativa.

Fonte: elaborado pelo autor.

Archer (2017) desenvolveu em seu projeto uma aplicação para apresentar uma versão holográfica da assistente virtual da Microsoft, a Cortana. Segundo o autor, esse avatar é o que ele imagina da Cortana ou o Google Home se essas assistentes utilizassem a coadjuvante holográfica da franquia de videogame Halo. O projeto utiliza um dispositivo Windows 10 nativo, com 4Gb de memória, e uma placa Arduino para controle das luzes da plataforma. Um monitor de LED USB projeta a animação para a pirâmide. A Figura 10 mostra o projeto construído.

Figura 10 - Conceito de holograma da Cortana

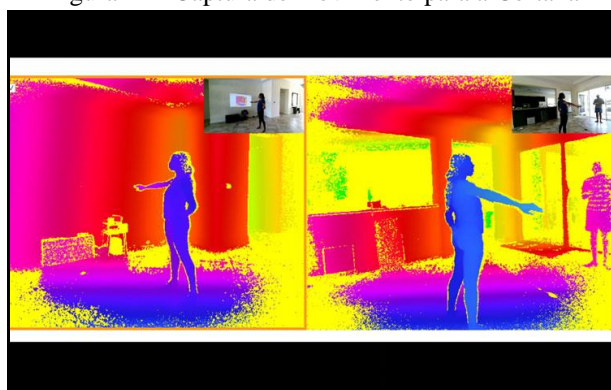


Fonte: Archer (2017).

O software responsável pelo controle da plataforma roda em duas partes. Uma delas é um aplicativo 3D desenvolvido em Unity que apresenta e anima a Cortana em três ângulos de câmera diferentes que se comunica com o segundo aplicativo, um serviço proxy que analisa os dados que são enviados e que são recebidos pela Cortana, renderiza o retorno HTML da assistente e retorna ao primeiro aplicativo para então ser apresentado. Também é empregada uma câmera que por meio de reconhecimento facial move a perspectiva de visão do aplicativo Unity, procurando deixar a avatar mais 3D.

Os movimentos de animação da Cortana foram capturados com o auxílio da esposa de Archer (2017), que se submeteu a várias tomadas de captura de movimento na sala de estar da casa do autor. Para a captura dos movimentos, foram utilizados dois dispositivos Kinect e a captura foi então inserida na animação da Cortana. A Figura 11 demonstra algumas cenas capturadas.

Figura 11 - Captura de movimento para a Cortana



Fonte: Archer (2017).

Segundo o autor, saber quando a Cortana nativa estava sendo exibida era um grande desafio, por se tratar de um aplicativo da UWP, ou Universal Windows Platform. Isso significa que seu design *sandbox* impede que sejam feitas chamadas de *handle* de janela, para verificar se um aplicativo está iniciado. O autor então solucionou o problema monitorando um único pixel da barra de tarefas do Windows para determinar quando a UI da Cortana estava presente. Por se tratar da Cortana nativa, este dispositivo pode ser usado não apenas para pesquisas de base de conhecimento, mas também para automação residencial e tarefas de música.

3 DESCRIÇÃO DA BIBLIOTECA

Esta seção tem por objetivo apresentar uma visão geral da funcionalidade da biblioteca desenvolvida e da aplicação de demonstração, bem como aspectos importantes do seu desenvolvimento. Esta seção está dividida em visão geral, implementação e operacionalidade.

3.1 VISÃO GERAL

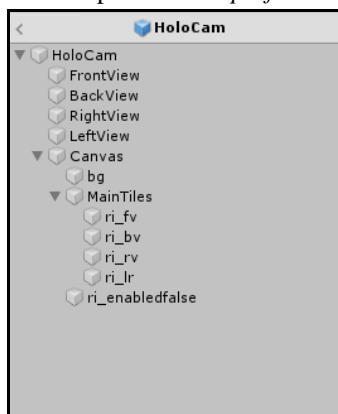
Nessa seção é abordado o objetivo principal da biblioteca de projeção e da aplicação de demonstração, seus requisitos e sua estrutura.

3.1.1 Visão geral da biblioteca

Esta biblioteca tem como principal objetivo facilitar a projeção de objetos na pirâmide holográfica. Ela disponibiliza ao usuário da Unity em formato de *prefab*, que consiste em quatro câmeras sendo uma câmera para cada face da pirâmide, um *canvas* com quatro imagens que irão receber o retorno das câmeras e uma imagem estática de posicionamento da pirâmide (Figura 12). A biblioteca disponibiliza uma forma de gerar os pontos de vista necessários para se obter uma projeção na pirâmide holográfica, sendo possível parametrizar um objeto alvo para a projeção, bem como a distância entre este alvo e as câmeras, e se a biblioteca está em formato de configuração. A biblioteca foi desenvolvida com base nos seguintes Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF):

- a biblioteca deve permitir a renderização dos quatro pontos de vista para projeção na pirâmide (RF);
- a biblioteca deve permitir a configuração de distância entre a câmera e o alvo da projeção (RF);
- a biblioteca deve conter uma tela de posicionamento da pirâmide holográfica (RF);
- a biblioteca deve ser desenvolvida em C# para o software Unity (RNF).

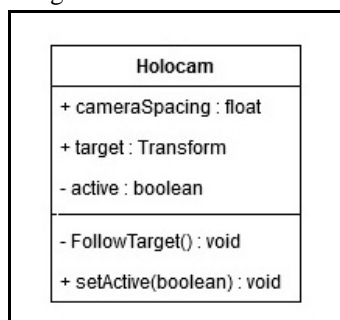
Figura 12 - Componentes do *prefab* de projeção



Fonte: elaborado pelo autor.

Para adquirir a projeção, a biblioteca utiliza o *script* cuja classe pode ser observada na Figura 13, responsável pelo posicionamento do *prefab* junto ao objeto alvo da projeção informando na variável *target* qual é o objeto alvo. Como a classe *Holocam* estende a classe *MonoBehaviour* própria do Unity, entende-se que os métodos *Start()* e *Update()* estão presentes também na classe *Holocam*, mas foram ocultos no diagrama.

Figura 13 - Classe *Holocam*



Fonte: elaborado pelo autor.

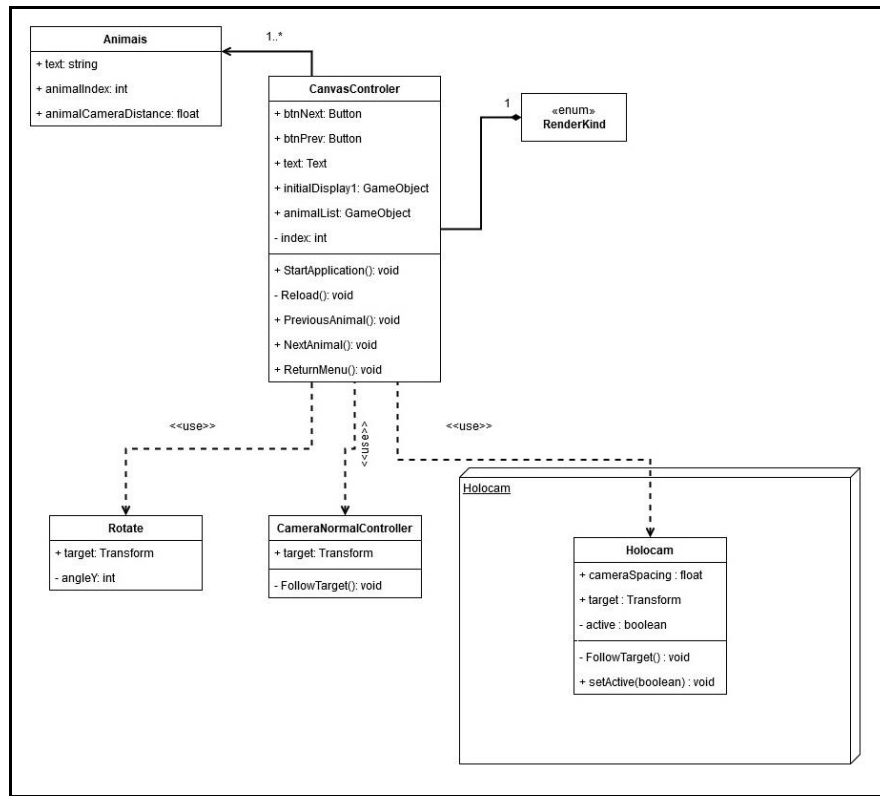
3.1.2 Visão geral da aplicação de demonstração

Foi desenvolvida em conjunto com a biblioteca uma aplicação de demonstração com o objetivo de comparar três modos de visualização diferentes, sendo eles o modo normal, a projeção na pirâmide holográfica utilizando a biblioteca desenvolvida e um modo de realidade aumentada (XR) utilizando um CardBoard. É disponibilizada uma lista com dez animais e o usuário pode avançar e retroceder na lista para visualizar o animal selecionado juntamente com um pequeno texto informativo sobre ele. A Figura 14 exibe o diagrama de classes dessa aplicação, utilizando também a biblioteca desenvolvida. A aplicação foi desenvolvida observando os seguintes RFs e RNFs:

- a aplicação deve permitir a visualização normal do modelo (RF);
- a aplicação deve permitir a visualização com a pirâmide holográfica (RF);
- a aplicação deve permitir a visualização com o CardBoard (RF);
- a aplicação deve permitir a navegação na lista de animais (RF);
- a aplicação deverá utilizar dois monitores (RNF);

- f) a aplicação deve utilizar a biblioteca desenvolvida (RNF);
g) a aplicação deve ser implementada em Unity (RNF).

Figura 14 - Diagrama de classes



Fonte: elaborado pelo autor.

Da mesma forma que a biblioteca de projeção, entende-se que os métodos `Update()` e `Start()` existem em todas as classes do diagrama, mas foram ocultados, exceto a classe **Animais** que por se tratar de um `ScriptableObject`, classe utilizada para criar objetos com valores padrão dentro do Unity, não possui métodos `Update()` e `Start()`. Nesse caso, são informados na classe **Animais** os valores do índice do animal dentro da lista de animais que existe em todas as cenas como um `GameObject` com os modelos dos animais adicionados dentro dele conforme a Figura 15. Também nessa classe é descrito o texto informativo sobre o animal e o espaçamento entre o modelo alvo e a câmera da cena.

Figura 15 - Lista de animais



Fonte: elaborado pelo autor.

A classe **CanvasController** é responsável por gerenciar a interação do usuário com as cenas, controlando ações como a navegação na lista de animais e voltar ao menu inicial. Para a navegação existem os métodos `NextAnimal()` e `PreviousAnimal()`, que avançam e retrocedem na lista de animais respectivamente, ambos métodos utilizando a função `Reload()` que é responsável por atualizar o animal em exibição e o texto informativo. Essa classe também utiliza o enum `RenderKind` para controlar qual *script* de projeção deve ser utilizado, sendo os valores possíveis `NORMAL`, `HOLOGRAPHIC` e `XR`. Quando o valor informado for `NORMAL` ou `XR` a classe **CanvasController** utiliza o *script* **CameraNormalController** para fazer o posicionamento da câmera e o *script* **Rotate** para rotacionar o animal selecionado no seu eixo Y. Quando o valor de `RenderKind` for `HOLOGRAPHIC`, a classe utiliza a biblioteca **Holocam** para gerar a projeção na pirâmide holográfica, não sendo necessário o *script* **Rotate**.

3.2 IMPLEMENTAÇÃO

Nessa seção são apresentadas informações técnicas sobre o desenvolvimento da biblioteca de projeção e da aplicação de demonstração. Também é apresentada a montagem da pirâmide holográfica com informações relevantes sobre sua estrutura.

3.2.1 Montagem da pirâmide

Em primeiro momento foi optado por se produzir um protótipo da pirâmide holográfica para a tela de uma *smartphone* utilizando acrílico. O material foi selecionado pois é de fácil acesso, sendo encontrado em capas transparentes de CDs. Foram recortadas quatro faces com as dimensões de 3 centímetros (cm) de topo, 6 cm de base e 5 cm de altura, sendo que mesmo com a pirâmide invertida a nomenclatura das dimensões não é alterada e o topo da pirâmide está em contato com a tela do *smartphone*. Optou-se por fixar a pirâmide com fita adesiva para eventuais ajustes na estrutura da pirâmide. A Figura 16 demonstra um teste feito utilizando uma animação do Pokémon Charmander encontrada no YouTube.

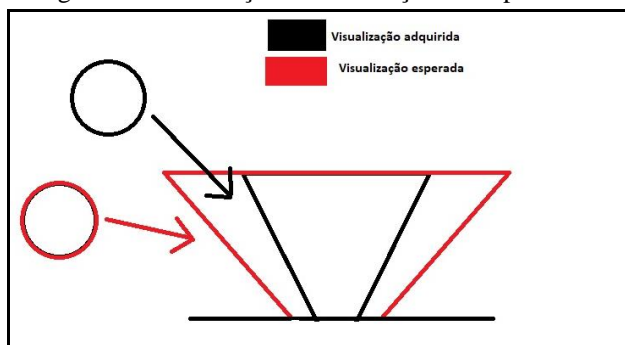
Figura 16 - Teste utilizando a pirâmide de acrílico



Fonte: elaborado pelo autor.

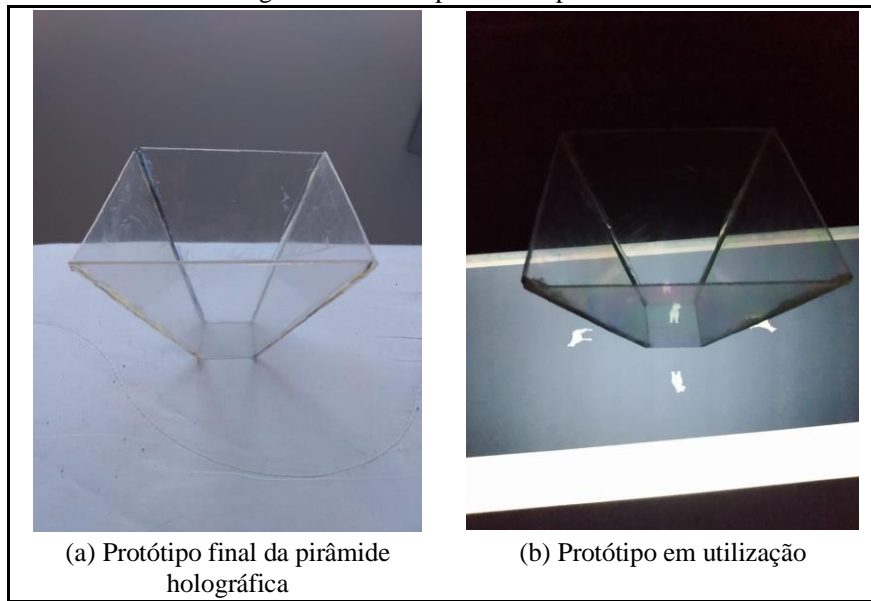
Após a finalização deste teste, passou-se então à construção de uma versão maior da pirâmide. Seguindo os cálculos de Hoffman (2017) para a montagem correta da pirâmide para o tamanho de tela de 15 polegadas, observou-se que o material não iria ser suficiente para as quatro faces, então foi diminuído o tamanho da base da face, consequentemente aumentando o ângulo em relação à tela. Essa alteração faz com que o usuário tenha que olhar para a face da pirâmide mais do alto, como demonstra a Figura 17. O protótipo final da pirâmide pode ser visto sozinho (Figura 18-a) e em utilização (Figura 18-b).

Figura 17 - Diferença de visualização entre pirâmides



Fonte: elaborado pelo autor.

Figura 18 - Protótipo final da pirâmide

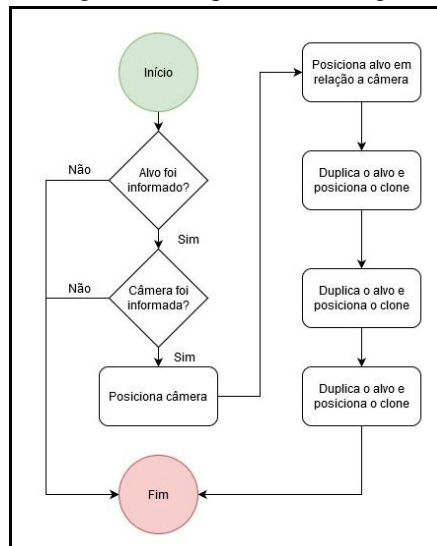


Fonte: elaborado pelo autor.

3.2.2 Biblioteca de projeção holográfica

Para o desenvolvimento da biblioteca de projeção holográfica, inicialmente foi optado por um código que seria adicionado à cena em que o objeto a ser projetado se encontra. O código seria responsável por clonar o objeto três vezes e posicionar os objetos resultantes e a câmera da cena (Figura 19) para se adquirir os quatro pontos de vista necessários para a projeção, conforme demonstra o Quadro 4.

Figura 19 - Diagrama de sequência do código de clonagem



Fonte: elaborado pelo autor.

Quadro 4 - Script de clonagem

```
void Start()
{
    if(target != null && cam != null)
    {
        cam.transform.position = new Vector3(0, 0.5f, -1);
        cam.transform.eulerAngles = new Vector3(0, 0, 0);
        cam.transform.localScale = new Vector3(1, 1, 1);

        target.transform.position = new Vector3(cam.transform.position.x,
        cam.transform.position.y+0.3f, cam.transform.position.z+3f);
        target.transform.eulerAngles = cam.transform.eulerAngles;

        copy_1 = Instantiate(target);
        copy_1.transform.position = new Vector3(cam.transform.position.x,
        cam.transform.position.y-0.3f, target.transform.position.z);
        copy_1.transform.eulerAngles = new Vector3(180, 0, 0);
        copy_1.transform.localScale = target.transform.localScale;

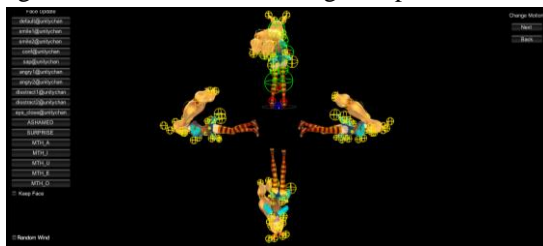
        copy_2 = Instantiate(target);
        copy_2.transform.position = new Vector3(cam.transform.position.x+0.3f,
        cam.transform.position.y, target.transform.position.z);
        copy_2.transform.eulerAngles = new Vector3(90, 0, -90);
        copy_2.transform.localScale = target.transform.localScale;

        copy_3 = Instantiate(target);
        copy_3.transform.position = new Vector3(cam.transform.position.x-0.3f,
        cam.transform.position.y, target.transform.position.z);
        copy_3.transform.eulerAngles = new Vector3(90, 0, 90);
        copy_3.transform.localScale = target.transform.localScale;
    }
}
```

Fonte: elaborado pelo autor.

O posicionamento dos objetos se dá da seguinte forma: a câmera informada na variável `cam` é posicionada nas coordenadas $X = 0$, $Y = 0.5$ e $Z = -1$ na cena, e sua rotação é zerada em todos os eixos. Após isto, o alvo informado na variável `target` é posicionado com $X = X$ de `cam`, $Y = Y$ de `cam` + 0.3 e $Z = Z$ de `cam` + 3, e sua rotação se iguala a da câmera. Feito esse passo, o primeiro clone é instanciado e posicionado em $X = X$ de `cam`, $Y = Y$ de `cam` - 0.3 e $Z = Z$ de `target`. O objeto clonado também é rotacionado 180° em X. Para o segundo clone, sua posição fica em $X = X$ de `cam` + 0.3, $Y = Y$ de `cam` e $Z = Z$ de `target`, e ele é rotacionado 90° em X e -90° em Z. O último clone é então instanciado, posicionado em $X = X$ de `cam` - 0.3, $Y = Y$ de `cam` e $Z = Z$ de `target`, e é rotacionado 90° em X e Z. O resultado desta operação pode ser visto na Figura 20.

Figura 20 - Resultado da clonagem e posicionamento

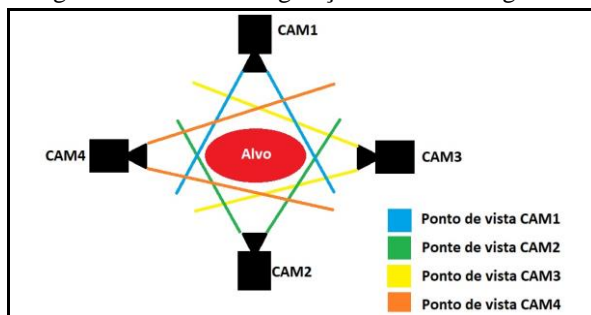


Fonte: elaborado pelo autor.

Porém, quando mais testes foram realizados com o código, o resultado não foi satisfatório, pois conforme foram adicionados mais objetos a cena, mais cópias do código eram requeridas, e os objetos eram adicionados todos na mesma posição, causando uma sobreposição. Foi pensado em uma forma de corrigir o problema utilizando mais variáveis na clonagem, para espaçamento entre objetos, mas isso não iria solucionar o fato de que quanto mais objetos para clonar, mais custoso se tornaria o trabalho de gerar a cena para projeção.

Diante deste resultado, foi iniciado o desenvolvimento de um segundo *asset*, utilizando uma técnica muito comum no desenvolvimento de minimaps em jogos de *first-player shooter* (FPS) e *role-playing game* (RPG), onde se captura o retorno de uma câmera na cena e direciona este resultado a uma imagem. Desta forma não são instanciados mais objetos, foi criado o *prefab* mencionado anteriormente. Para o posicionamento das câmeras foram utilizados os cálculos desenvolvidos na estratégia de clonagem, assim conseguindo os quatro pontos de visualização mais facilmente conforme a Figura 21.

Figura 21 - Modelo de geração de cena holográfica



Fonte: elaborado pelo autor.

O Quadro 5 mostra o método `FollowTarget`, responsável por posicionar o *prefab* junto ao objeto alvo da projeção, para que as câmeras capturem o objeto corretamente. Com a utilização da variável `cameraSpacing`, é possível se ter um modelo 3D que fique fora do *field of vision* (FOV) das câmeras na cena e afastar as câmeras dele, fazendo com que o objeto fique dentro do espaço correto para a projeção. O mesmo ocorre com um objeto pequeno, é possível diminuir a distância entre as câmeras e o objeto informando nessa variável um valor menor que zero, diminuindo a distância entre o `target` e as câmeras fazendo assim que a imagem aumente. Vale ressaltar que essa variável é utilizada para todas as câmeras, assim é necessário o desenvolvedor encontrar um valor para a variável.

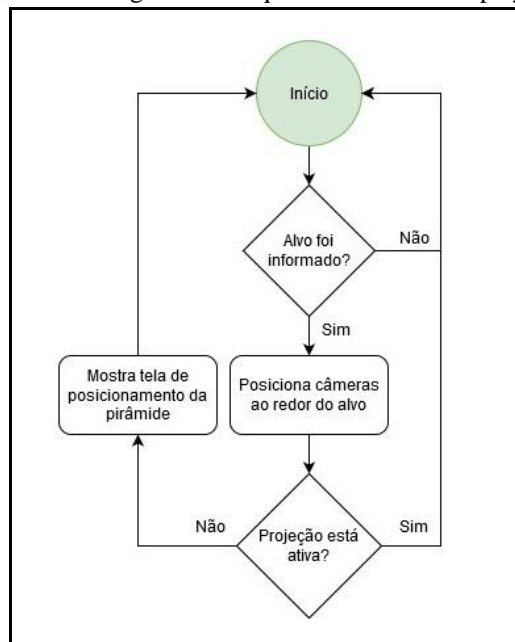
Quadro 5 - Método `FollowTarget`

```
void FollowTarget()
{
    for (int i = 0; i < this.gameObject.transform.childCount; i++)
    {
        GameObject child = this.gameObject.transform.GetChild(i).gameObject;
        float x, y, z;
        x = target.transform.position.x;
        y = target.transform.position.y;
        z = target.transform.position.z;
        if(child.tag == "FrontView")
        {
            child.transform.position = new Vector3(x, y, z + cameraSpacing);
        }
        else if(child.tag == "BackView")
        {
            child.transform.position = new Vector3(x, y, z - cameraSpacing);
        }
        else if(child.tag == "RightView")
        {
            child.transform.position = new Vector3(x - cameraSpacing, y, z);
        }
        else if(child.tag == "LeftView")
        {
            child.transform.position = new Vector3(x + cameraSpacing, y, z);
        }
        else if(child.tag == "Canvas")
        {
            for(int j = 0; j < child.transform.childCount; j++)
            {
                GameObject gc = child.transform.GetChild(j).gameObject;
                if(gc.tag == "MainTiles")
                {
                    gc.SetActive(active);
                }
                else if(gc.tag == "EnabledFalseTiles")
                {
                    gc.SetActive(!active);
                }
            }
        }
    }
}
```

Fonte: elaborado pelo autor.

Conforme demonstra a Figura 22, a biblioteca inicia verificando se existe um alvo informado na variável `target` e se este for o caso, o *prefab* é posicionado junto ao alvo. Como as câmeras tem sua posição fixa dentro do *prefab* de acordo com os cálculos adquiridos na estratégia de clonagem, a projeção tem dessa forma os pontos de vista corretos para a projeção, levando em conta o espaçamento entre o alvo e as câmeras que é informado pelo usuário na variável `cameraSpacing`. Caso contrário elas permanecem na posição original, sendo esta onde o usuário posicionou o *prefab* na cena do Unity. Caso a projeção não esteja habilitada, é demonstrada uma tela de posicionamento da pirâmide. Por se tratar de um método chamado na função `Update()`, não tem um final pois o método `Update()` é chamado a cada frame.

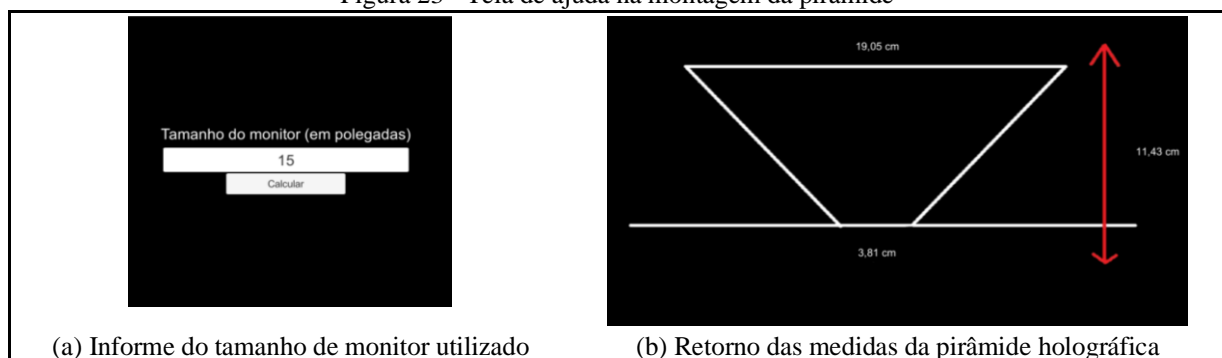
Figura 22 - Diagrama de sequência do modo de projeção



Fonte: elaborado pelo autor.

Foi desenvolvido em conjunto com o *prefab* uma cena de ajuda para a montagem correta da pirâmide, onde o usuário informa o tamanho do monitor utilizado para a projeção e são feitos os cálculos desenvolvidos por Hoffman (2017) para as medidas do topo, base e altura da pirâmide. A cena pode ser vista em utilização na Figura 23, onde é informado que se é utilizado um monitor de 15 polegadas (Figura 23-a) e são retornados os valores das medidas relativas (Figura 23-b). Como não se sabe qual o material utilizado pelo usuário e quais as dimensões em que o efeito de duplicata constatado por Schivani *et al.* (2018) acontecem, não foi informado nessa tela a necessidade de aplicação das películas automotivas.

Figura 23 - Tela de ajuda na montagem da pirâmide

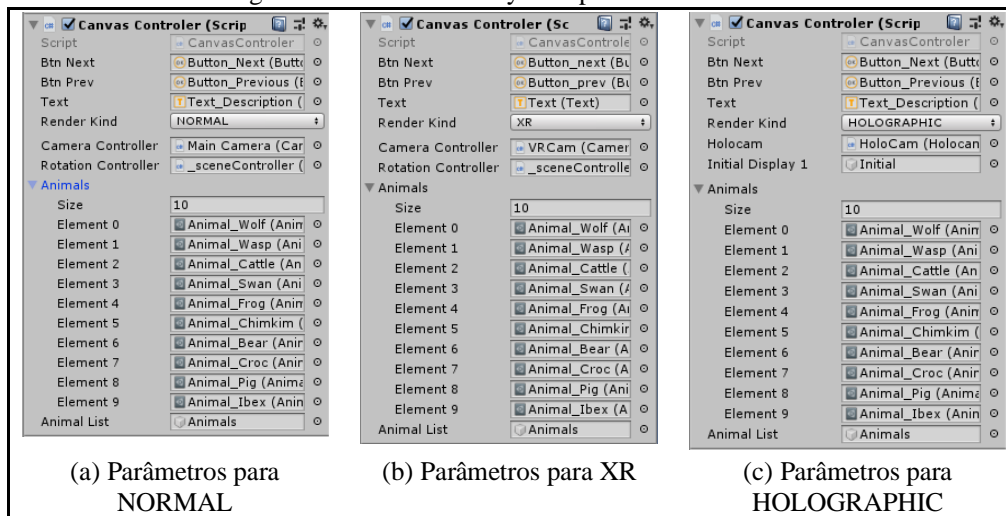


Fonte: elaborado pelo autor.

3.2.3 Aplicação de demonstração

Na implementação da aplicação de demonstração como pode ser visto na Figura 24, ao alterar o valor de `RenderKind` no script `CanvasController`, o editor do Unity altera as opções necessárias para a cena funcionar. A classe `CanvasController` utiliza o método `Reload()` para atualizar o animal projetado conforme pode ser observado no Quadro 6. Este método é responsável também por habilitar e desabilitar os botões de navegação de acordo com a variável `index`.

Figura 24 - Editor do Unity com parâmetros variáveis



Fonte: elaborado pelo autor.

Quadro 6 - Método Reload

```
private void Reload()
{
    btnPrev.enabled = (index == 0 ? false : true);
    btnNext.enabled = (index == animals.Length-1 ? false : true);

    text.text = animals[index].text;
    if (renderKind == RenderKind.HOLOGRAPHIC)
    {
        holocam.cameraSpacing =
animals[index].animalCameraDistance;
        holocam.target =
animalList.transform.GetChild(animals[index].animalIndex);
    }
    else
    {
        cameraController.target =
animalList.transform.GetChild(animals[index].animalIndex);
        rotationController.target =
animalList.transform.GetChild(animals[index].animalIndex);
    }
}
```

Fonte: elaborado pelo autor.

Como pode ser visto, o método `Reload()` indica o próximo alvo de projeção aos *scripts* de projeção `Holocam` e `cameraNormalController`, de acordo com o valor de `renderKind`. Assim, é possível utilizar a classe `CanvasController` em todas as cenas de projeção não sendo necessário um *script* de controle diferente para cada cena.

A classe `CameraNormalController` utilizada nas cenas de modo Normal e XR faz o distanciamento entre a câmera da cena e o alvo da projeção no método `FollowTarget()` com valores fixos conforme demonstrado no Quadro 7, pois não foi necessário um valor variável nesses modos de projeção sendo que os modelos ficaram bem colocados com os valores informados. O método `FollowTarget()` é chamado dentro do método `Update()` da classe, sendo executado a cada *frame*.

Quadro 7 - Método FollowTarget do script CameraNormalController

```
private void FollowTarget()
{
    float posY = target.transform.position.y + 1f;
    float posX = target.transform.position.x + 5f;
    float posZ = target.transform.position.z - 5f;

    this.transform.position = new Vector3(posX, posY, posZ);
    this.transform.eulerAngles = new Vector3(0, 310f, 0);
}
```

Fonte: elaborado pelo autor.

Assim como a classe `CameraNormalController`, a classe `Rotate` também é utilizada nas cenas de modo Normal e XR, não sendo necessário na cena de modo Holografia pois é possível ver o animal de todos os ângulos nesse modo. O script `Rotate` rotaciona o alvo informado na variável `target` no seu eixo Y um grau por *frame*, conforme pode ser observado no Quadro 8.

Quadro 8 - Método Update do script Rotate

```
void Update()
{
    if(target != null)
    {
        Vector3 rotation = new Vector3(target.eulerAngles.x,
angleY, target.eulerAngles.z);
        target.eulerAngles = rotation;
        angleY++;
    }
}
```

Fonte: elaborado pelo autor.

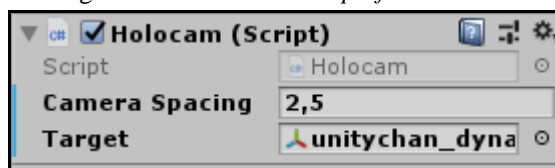
3.3 OPERACIONALIDADE

Nessa seção são informados os passos para utilizar a biblioteca desenvolvida e a aplicação de demonstração. Também são informados os requisitos para que a aplicação de demonstração funcione corretamente.

3.3.1 Biblioteca de projeção

Para utilizar a biblioteca desenvolvida é necessário importar o *asset* disponibilizado dentro do Unity. Após a importação, ficará disponível o *prefab* `Holocam` que pode ser então inserido na cena desejada. Com o *prefab* inserido na cena, é necessário informar um alvo para a projeção na variável `target` e uma distância entre o alvo e as câmeras na variável `cameraSpacing`, como demonstra a Figura 25.

Figura 25 - Parâmetros do *prefab* Holocam



Fonte: elaborado pelo autor.

Após serem informados os dois parâmetros, ao renderizar a cena a projeção está ativa e funcionando. Para visualizar a imagem de posicionamento da pirâmide se deve chamar o método `SetActive()` passando como parâmetro um valor booleano `false` e para retomar a projeção deve-se chamar o mesmo método passando um valor `true`.

Para visualizar a rotina de cálculo da pirâmide holográfica, é disponibilizada junto ao *prefab* uma cena que pode ser chamada a qualquer momento, sendo necessário informar ao Unity que a cena vai ser utilizada nas suas *build settings*.

3.3.2 Aplicação de demonstração

Para a utilização desta aplicação são necessários dois monitores, um para o controle da cena e um para a projeção holográfica. Assim os controles da cena não interferem na projeção da pirâmide, sendo que quando não se está no modo holografia, o segundo monitor se mantém em um “estado neutro”, onde a tela fica escura. A Figura 26 ilustra o menu principal da aplicação.

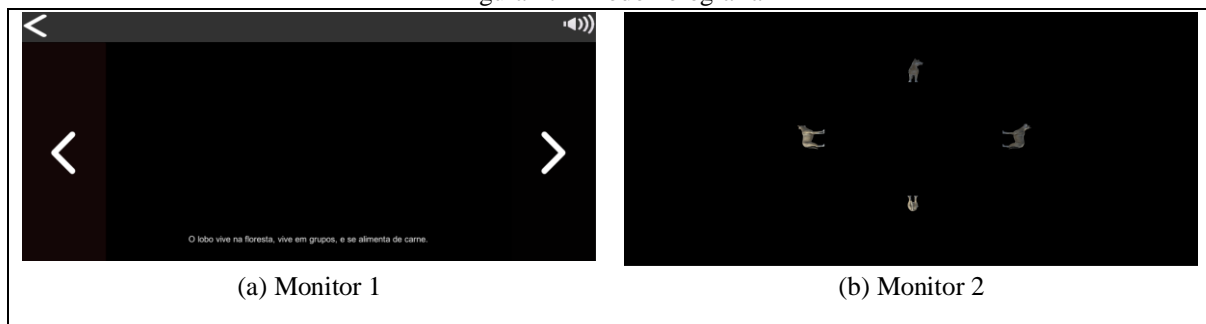
Figura 26 - Menu inicial da aplicação de teste



Fonte: elaborado pelo autor.

A opção *Holografia* do menu principal disponibiliza uma cena holográfica utilizando a biblioteca desenvolvida para gerar a projeção do animal selecionado no monitor secundário (Figura 27-b), e os controles da cena no monitor primário (Figura 27-a).

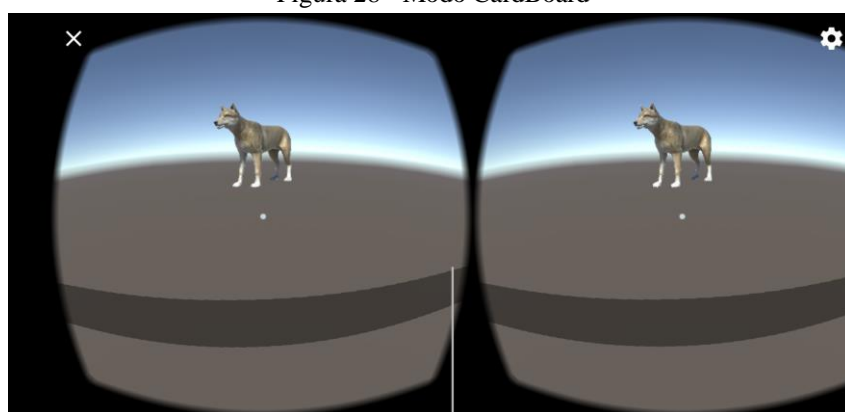
Figura 27 - Modo holografia



Fonte: elaborado pelo autor.

A opção de *Cardboard* do menu principal exige a utilização de um dispositivo móvel compatível com o aplicativo Google Cardboard. Nesse modo de visualização o usuário utiliza um óculos de realidade aumentada para escolher um animal e visualizá-lo, conforme demonstra a Figura 28. Como a posição da câmera nesse modo de visualização é fixa, optou-se por rotacionar o animal selecionado para o usuário poder visualizá-lo de todos os ângulos.

Figura 28 - Modo CardBoard



Fonte: elaborado pelo autor.

A opção *Normal* do menu principal apresenta uma cena simples com o modelo do animal selecionado exibido no centro da tela. Nesta opção é utilizado apenas o monitor primário, os controles e o texto informativo estão situados na mesma tela que o animal. Também foi optado por rotacionar o animal nesta cena, pois a câmera é fixa e o distanciamento da câmera também é controlado via *script*. A Figura 29 demonstra a cena normal em utilização.

Figura 29 - Cena normal da aplicação de teste



Fonte: elaborado pelo autor.

4 RESULTADOS

A seção de resultados foi dividida em três partes. A primeira trata dos testes com a parte física, ou seja, a pirâmide holográfica. A segunda engloba testes feitos com a biblioteca de projeção. Após isso são apresentados os testes da aplicação que utiliza a biblioteca de projeção desenvolvida, comparando com os outros modos de visualização e finalmente se tem uma comparação com os trabalhos correlatos.

4.1 TESTES DA PIRÂMIDE HOLOGRAFICA

Antes de começar o desenvolvimento da biblioteca e após a pirâmide estar montada, foi testada a capacidade de se visualizar o objeto na pirâmide em ambientes diferentes. Foram aplicados testes com três usuário em três cenários, medindo a luminosidade em lumens (lm), utilizando a pirâmide posicionada sobre um *smartphone* com o brilho da tela no máximo, e em seguida os usuários foram entrevistados remotamente. O primeiro ambiente teve sua iluminação medida a 1.384 lm, o segundo ambiente a 368 lm e o terceiro e último ambiente a 2 lm. Para realizar os testes, foi utilizada a imagem do Pokemon Charmander vista na Figura 16. Os usuários testados para validação afirmaram que o objeto foi mais visível no ambiente com 2 lm, confirmando que como a pirâmide holográfica utiliza os mesmos conceitos de reflexão da luz que a técnica de Pepper, a visibilidade do objeto na pirâmide depende da intensidade de brilho da tela em que a pirâmide está posicionada da mesma forma que depende da iluminação externa.

Também foram testadas as cores mais visíveis na pirâmide, mostrando aos mesmos usuários uma aplicação em Unity que exibe um cubo mudando de cor na cena. Esses testes foram inconclusivos, pois para um dos usuários o espectro de cores mais visível foram as cores mais claras independente se fossem cores quentes, frias, primarias, secundarias, entre outras. Já para outros dois usuários as cores mais visíveis estavam dentro do grupo de cores quentes, independente da intensidade. Concluiu-se que pessoas diferentes terão resultados diferentes nesta fase do teste, pois depende da visão de cada um.

4.2 TESTES DA BIBLIOTECA

Estes testes foram realizados em um notebook com um processador Intel Core i3 6006U de 2Ghz, 4Gb de memória RAM, sem placa gráfica e com sistema operacional Windows 10. Os testes possuem um teor técnico fazendo um comparativo entre os dois *assets* produzidos.

A princípio foi testada a capacidade de clonar objetos na cena com a estratégia de clonagem, onde foi constatado que quando se tem apenas um objeto estático não ocorrem dificuldades em gerar a cena, porém quanto mais objetos para serem clonados mais demorada se torna a operação, demonstrado na Tabela 1. Tendo em vista que são gerados três novos objetos de cada alvo e é necessário ressaltar que os objetos são todos posicionados no mesmo local causando assim uma sobreposição.

Tabela 1 - Tempo de renderização das estratégias

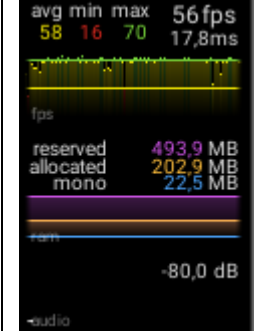
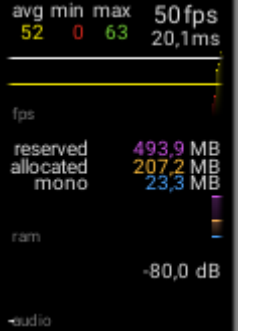
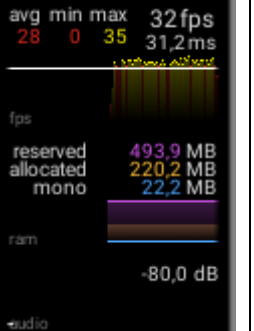
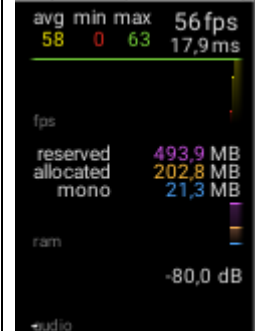
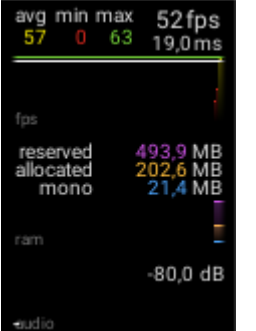
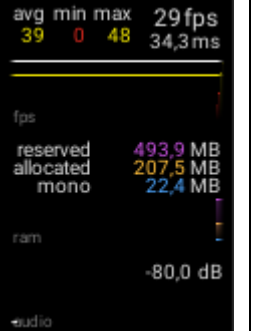
Número de objetos	Tempo da clonagem (segundos)	Tempo do <i>prefab</i> (segundos)
1	0,0199244	0,003307581
4	0,08895111	0,003201962
10	0,21444778	0,003155708

Fonte: elaborado pelo autor.

Conforme a Tabela 1, pode-se observar que o *prefab* de projeção holográfica tem um tempo de renderização constante, diferente da estratégia de clonagem pois não é necessário clonar os objetos para adquirir uma projeção. Considerando n como o número de objetos na cena, o script de clonagem desenvolvido tem sua curva de complexidade linear ($O(n)$), pois o custo da clonagem aumenta junto com o número de objetos a serem clonados. Já o *prefab* desenvolvido tem sua curva de complexidade constante ($O(1)$), pois não há custo de clonagem, logo o custo da renderização do *asset* tem uma variação de tempo mínima, mas não depende da quantidade de objetos em cena.

Também foram realizados testes para averiguar o custo de memória de cada estratégia de projeção, onde o *prefab* demonstrou um melhor desempenho conforme observado no Quadro 9, onde a primeira linha das imagens ilustram o desempenho da estratégia de clonagem e a segunda linha demonstra o desempenho do *prefab* desenvolvido. Como pode ser observado, existe uma diferença de 12.7Mb de memória alocada à cena entre as estratégias, confirmando assim que o *prefab* tem um desempenho melhor que a estratégia de clonagem. É possível notar também que quanto mais objetos na cena, mais diminui o FPS independente de qual estratégia for adotada. Esse comportamento é próprio do Unity.

Quadro 9 - Desempenho das estratégias

		
		
(a) Testes realizados com um objeto	(b) Testes realizados com quatro objetos	(c) Testes realizados com dez objetos

Fonte: elaborado pelo autor.

4.3 TESTES FINAIS

Para esses testes, um usuário utilizou a aplicação de demonstração com os três modos de projeção, e em seguida o mesmo foi entrevistado. Segundo o usuário, em relação aos modos de visualização normal e Cardboard, vê-se que a projeção em pirâmide holográfica é mais imersiva, pois o usuário teve a possibilidade de andar em volta do projetor. Como foi demonstrado é possível deixar o controle da cena em outra tela, não interferindo na projeção holográfica.

O modo de exibição normal foi apontado pelo usuário como desinteressante, por se tratar de objetos com volume exibidos em um plano 2D, causando uma impressão falsa de 3D. Segundo relato do usuário, é possível perceber a profundidade do objeto porém ela se torna superficial, se perdendo no plano 2D.

Já o modo de exibição Cardboard foi avaliado pelo usuário como mais interessante que o modo normal, porém para este usuário em questão, o modo holográfico se demonstrou mais interessante por permitir a visualização de vários ângulos ao se mover em torno da pirâmide. Para o usuário testador não é interessante apenas mover a cabeça como é feito no modo Cardboard para visualizar a cena.

4.4 COMPARAÇÕES COM TRABALHOS CORRELATOS

Com o intuito de gerar cenas de projeção na pirâmide holográfica, nenhum dos trabalhos correlatos utilizou uma biblioteca para auxiliar no desenvolvimento da cena. Hoffman (2018) e Archer (2017) duplicaram os objetos e os

posicionou na cena, entretanto não foram encontrados dados técnicos sobre o produto comercializado pela PixelSav (2015).

Em comparação com os trabalhos correlatos, exceto o totem holográfico comercializado pela PixelSav (2015), vê-se uma facilidade na criação de cenas de projeção na pirâmide holográfica. Não se faz necessário a clonagem e posicionamento manual dos objetos, ou da utilização da técnica de minimapa pura, já que o *prefab* desenvolvido faz automaticamente esse processo. Apenas é necessário importar a biblioteca e utilizá-la na cena desejada.

5 CONCLUSÕES

Apesar do baixo número de testes realizados com usuários, conclui-se que a biblioteca cumpre sua proposta, pois é possível gerar uma cena com os pontos de vista necessários para a pirâmide holográfica, sem a necessidade de copiar o objeto, colar e posicioná-lo na cena manualmente. Como provado, a biblioteca economiza recursos de tempo e memória ao gerar uma cena de projeção na pirâmide holográfica, pois não se faz necessário a duplicação do objeto para se adquirir os pontos de vista requeridos.

Para estender o desenvolvimento da biblioteca é possível deixar parametrizável quais faces da pirâmide podem ser visualizadas e quais devem ser ocultas. Também deixar visível o cenário por trás do objeto alvo da projeção, permitir remover ou adicionar câmeras a biblioteca e aumentar o número de alvos para a projeção. Outra sugestão seria implementar uma aplicação que utilize a biblioteca desenvolvida para gerar uma cena de projeção na pirâmide holográfica e aplicar alguma interação, similar ao que foi comercializado pela PixelSav (2015).

Por ter um baixo número de testes com usuário, sugere-se executar novos testes com mais usuários, englobando também usuários com problemas de visão, como daltonismo por exemplo, para averiguar como o desempenho da pirâmide é afetado. Como as opiniões mudam de pessoa para pessoa, é válido também ampliar o teste feito com a aplicação de demonstração dos três modos de exibição para que se possa analisar melhor os resultados obtidos.

REFERÊNCIAS

- ARCHER, J. **Holographic Cortana Appliance: Working Concept**. [S. l.], [2017]. Disponível em <http://untitled.com/blog/cortana-hologram-working-concept>. Acesso em 27 out. 2019.
- FERREIRA, C. LOPES, D. **Holografia**. 2017. 8 f. Artigo – Instituto Superior Técnico, Lisboa.
- GABOR, D. **Holography**. 1971. 35 f. Palestra (Nobel de Física) - Imperial College of Science and Technology, Londres.
- HOFFMAN, B. V. **Um estudo sobre a holografia aplicada a visualização do eclipse solar e lunar**. 2018. 61 f. Trabalho de Conclusão de Curso (Bacharel em Tecnologias Digitais) – Universidade de Caxias do Sul, Caxias do Sul.
- MEDEIROS, A. A história e a física do Fantasma de Pepper. **Caderno brasileiro de ensino de física**, UFSC, v. 23, n. 3, p. 329-345, dez, 2006.
- PIXELSAV. **Holografia: Pokemon**. Curitiba, 2015. Disponível em: http://www.pixelsav.com.br/projetos/pokemon_pixel/. Acesso em: 20 ago. 2019.
- REBORDÃO, J. M. Holografia: Física e Aplicações, **Colóquio/Ciências Revista de Cultura Científica**, Lisboa, n. 4, p.18-34, 1989.
- SCHIVANI, M. SOUZA, G. F. PEREIRA, E. Pirâmide “holográfica”: erros conceituais e potencial didático, **Revista Brasileira de Ensino de Física**, [S. l.], v. 80, n. 2, nov, 2018.
- UNITY, **Unity User Manual (2019.2)**, [S. l.], [2019]. Disponível em <https://docs.unity3d.com/Manual/index.html>. Acesso em 20 out. 2019.