

ATENÇÃO: aqui constam somente as páginas que tinham alguma anotação na revisão.

USO DA REALIDADE AUMENTADA COM MARCADORES DINÂMICOS

Everton da Silva, 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

evertonslv.silva@gmail.com, dalton@furb.br

Resumo: Esse artigo descreve o processo de desenvolvimento de um aplicativo que utiliza realidade aumentada e tem como objetivo criar, detectar e apresentar objetos 3D utilizando marcadores pré-definidos ou dinâmicos, permitindo que qualquer pessoa mesmo sem muito conhecimento tecnológico possa criar uma cena e utilizar realidade aumentada. O aplicativo foi desenvolvido através da plataforma Unity utilizando a linguagem de programação C# para escrever os scripts através da ferramenta Visual Studio Community, em conjunto com as bibliotecas OpenCV para Unity. O aplicativo permite que o usuário importe qualquer arquivo 3D (.fbx, .blender, .prefab e .obj) e em seguida rotacione esse objeto na posição desejada, após isso será criado as bordas desse objeto e sobreposto na tela do dispositivo permitindo que o usuário faça um desenho em uma folha de papel tendo como base o próprio objeto que ele selecionou. Foram realizados testes de usabilidade no aplicativo e foi encontrado algumas limitações na comparação do desenho do usuário com as bordas do objeto selecionado.

1
Testasse estas importações com estes formatos?

Palavras-chave: Marcadores dinâmicos. OpenCV for Unity. Realidade aumentada. OpenCV. Unity.

1 INTRODUÇÃO

Desde os primórdios da informática o homem busca diferentes maneiras de inovar a Interação Homem-Máquina. No início os computadores eram operados manualmente sem a utilização de painéis ou display. Com o passar dos anos foram sendo desenvolvidos monitores com uma interface mais amigável, sistemas operacionais baseados em interface gráfica, mouses e teclados. Essa evolução segue até os dias de hoje e um simples computador pode ser carregado na palma da mão (COSTA et al., 2011, p. 11).

Com esses avanços tecnológicos, surgiu a Realidade Aumentada (RA), permitindo a sobreposição de objetos e ambientes virtuais com o ambiente físico através de algum dispositivo tecnológico. A RA teve sua primeira aparição na década de 90, porém, ficou mais acessível no início dos anos 2000 com a convergência de técnicas de visão computacional, software e dispositivos com um melhor custo-benefício (KIRNER; SISCOUTTO, 2007, p. 5).

Em geral, a RA funciona através de marcadores, e estes são configurados e impressos com intuito da aplicação reconhecê-los e detectá-los, obtendo informações necessárias para identificar o que e onde gerar. Esta técnica é conhecida como Marker Based Tracking (rastreamento baseado em marcações) (HESS, 2011, p. 19).

Outra técnica adotada para os recursos da RA é o Markerless Tracking (rastreamento sem marcações), considerada a mais ideal, já que não necessita de marcadores impressos (BIMBER; RASKAR, 2005 apud HESS, 2011, p. 19). Realidade Aumentada sem marcadores do inglês Markerless Augmented Reality (MAR) é uma técnica baseada no reconhecimento de objetos do mundo real. Os sistemas baseados em MAR podem usar imagens e objetos reais para apresentar os efeitos da RA. MAR utiliza algoritmos de reconhecimento de imagens para detecção de objetos, ao contrário dos marcadores que tem sua estrutura fixa e conhecida (OZLU, 2018).

Segundo Teichrieb (2007 apud KUMAGAI, 2011, pg. 17), qualquer parte da cena real pode ser utilizado como marcador, podendo ser rastreada a posição do objeto. Teichrieb ainda afirma que os sistemas MAR possuem algumas vantagens por conter rastreadores especializados e robustos, além de possibilitar extrair características da cena real, porém para realizar tais ações pode ser complexo e apresentar restrições.

Uma biblioteca utilizada na RA é o OpenCV, uma biblioteca de visão computacional e aprendizado de máquina. O OpenCV tem integração com ArUco uma biblioteca utilizada para detecção de marcadores (ARUCO, 2020). O OpenCV disponibiliza alguns algoritmos de detecção de objetos como, Homografia e Transformação de Perspectiva que detecta pontos de semelhança entre duas imagens e que podem auxiliar no desenvolvimento do MAR.

Diante do exposto, este trabalho apresenta o desenvolvimento de uma aplicação utilizando os recursos da RA baseado em marcadores dinâmicos, que não necessita de marcadores pré-definidos na cena. Com intuito de melhorar a interação com o usuário, permitindo que qualquer pessoa, mesmo sem muito conhecimento tecnológico consiga criar ou adquirir da internet um objeto 3D e adicionar a uma cena do aplicativo e utilizar a RA.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os aspectos da fundamentação teórica utilizados na construção deste trabalho. Na primeira subseção desta seção são apresentados os conceitos utilizados como base para o desenvolvimento da aplicação. Na segunda subseção são apresentados os trabalhos relacionados à ferramenta desenvolvida.

2.1 OPENCV

O Open Source Computer Vision (OpenCV) é uma biblioteca de software de visão computacional e aprendizado de máquina de código aberto, que possui mais de 2.500 algoritmos. Estes algoritmos podem ser utilizados para detectar e reconhecer rostos, identificar objetos e classificar ações humanas em vídeos. Também permite rastrear movimentos de câmeras, rastrear objetos em movimento, extrair modelos 3D de objetos, produzir nuvens de pontos 3D de câmeras estéreo, juntar imagens para produzir uma alta resolução imagem de uma cena inteira. Bem como encontrar imagens semelhantes em um banco de dados, remover olhos vermelhos de imagens, seguir os movimentos dos olhos, reconhecer cenários e estabelecer marcadores para sobrepor com realidade aumentada, entre outros (OpenCV, 2019).

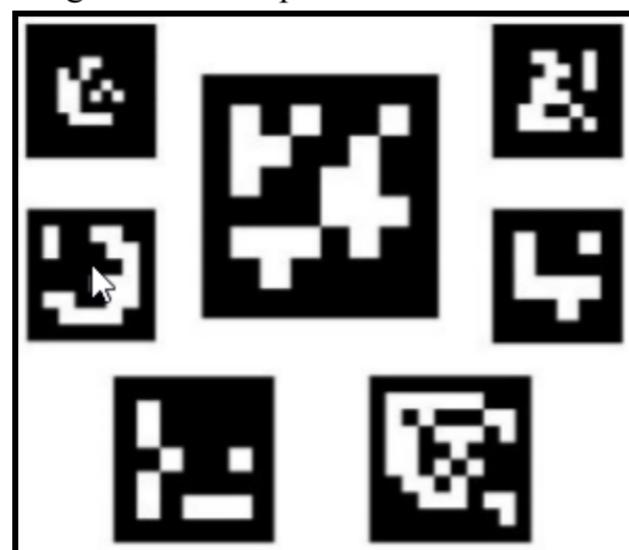
A biblioteca OpenCV é escrita nativamente em C++ e possui interfaces para C++, Python, Java e MATLAB. O OpenCV suporta as plataformas, Windows, Linux, Android e MacOS (OpenCV, 2019). OpenCV é compatível também com Unity, porém, para conseguir tal compatibilidade é necessário declarar todas as funções C/C++ para exportação e em seguida compilar todo o código do OpenCV como um pacote de biblioteca e importá-lo no projeto como um *plug-in* (ENOX SOFTWARE, 2020).

A empresa Enox Software desenvolveu o *plug-in* OpenCV for Unity para prover eficiência computacional, sobretudo em aplicações em tempo real. O *plug-in* disponibiliza uma API para C# baseada na API para Java disponibilizada pelo próprio OpenCV. Para adquirir a API OpenCV for Unity é necessário fazer o download na loja *assets* do Unity e comprar a licença da mesma (ENOX SOFTWARE, 2020).

2.2 ARUCO (REALIDADE AUMENTADA COM MARCADORES)

Augmented Reality University of Cordoba (ArUco) é uma biblioteca Open Source escrita em C++ e com integração com OpenCV, OpenGL e OGRE. A ArUco é responsável pela execução de rotinas de detecção de marcadores. A ArUco gera os marcadores quadrados sintéticos compostos por uma borda preta larga e com uma matriz binária que determina um identificador único dentro deles. A borda preta facilita sua rápida detecção na imagem e a codificação binária permite sua identificação. O tamanho do marcador determina o tamanho da matriz interna, por exemplo, um marcador 4x4 é composto por 16 bits (ARUCO, 2020). A Figura 1 mostra um exemplo de marcadores ArUco.

Figura 1 - Exemplo marcadores ArUco



Fonte: Aruco (2020).

As principais características do ArUco são: detectar marcadores com uma única linha de código; detectar bibliotecas como: AprilTag, ArToolKit, ArToolKit+, ARTAG, CHILITAGS; mais rápido que qualquer outra biblioteca para detecção de marcadores; ser multiplataforma (Windows, Linux, Mac OS, Android); e poucas dependências (AVA, 2018).

2.3 HOMOGRAFIA E TRANSFORMAÇÃO DE PERSPECTIVA

OpenCV disponibiliza alguns algoritmos para processamento de imagens, tais como, Homografia e Transformação de Perspectiva para detecção de objetos em tempo real.

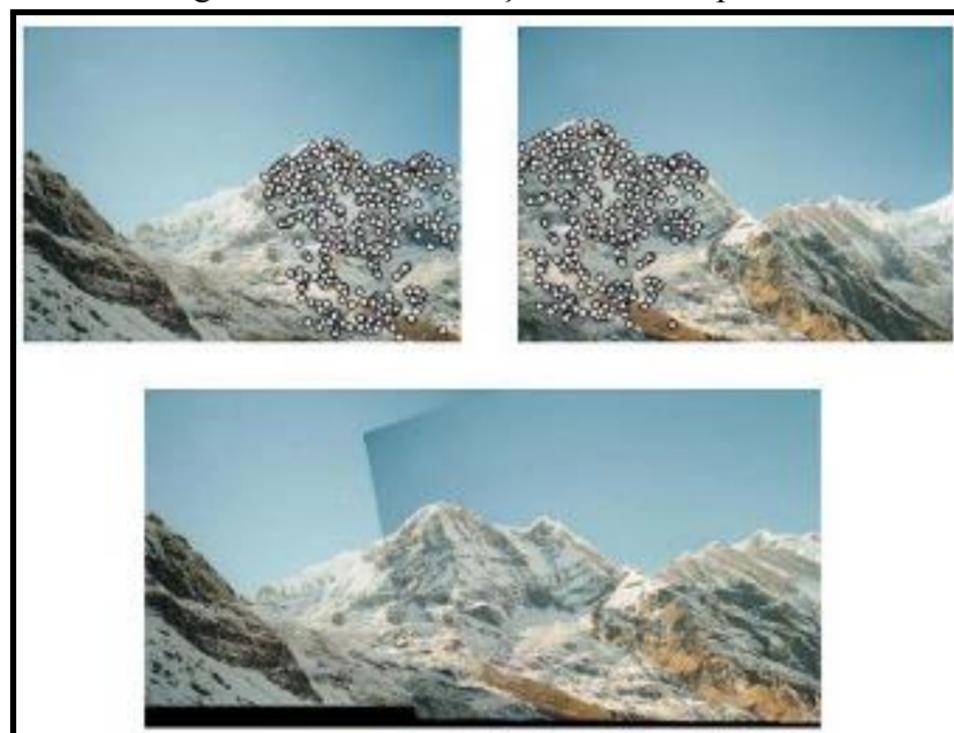
Homografia é uma relação plana que transforma pontos de um plano para outro. É uma matriz 3 por 3 que transforma vetores tridimensionais que representam os pontos 2D no plano. Esses vetores são chamados de coordenadas homogêneas (PERI, 2017).

source

Já coordenadas homogêneas são coordenadas projetivas que representam pontos dentro da visão computacional. Como existe uma conversão 3D para 2D em uma foto, a escala de profundidade é perdida. Com isso, uma quantidade infinita de pontos 3D é projetada para o mesmo ponto 2D, tornando as coordenadas homogêneas muito versátil na descrição do raio de possibilidade, uma que são semelhantes em escala (PERI, 2017).

O algoritmo da homografia é muito simples em relação a outros algoritmos por ser direto e intuitivo, porém, só é possível utilizar homografia quando a coordenada Z for igual a 0. Pois a homografia só funciona em cenários planos, caso contrário, outras abordagens são necessárias. Desta forma o algoritmo se torna inútil caso o alvo desejado saia de vista da câmera, sendo necessário movimentar a câmera, até que ela possa olhar o tempo todo para o alvo (PERI, 2017). A Figura 2 mostra um exemplo de como a homografia relaciona a transformação entre dois planos.

Figura 2 – Transformação entre dois planos



Fonte: OpenCV (2020).

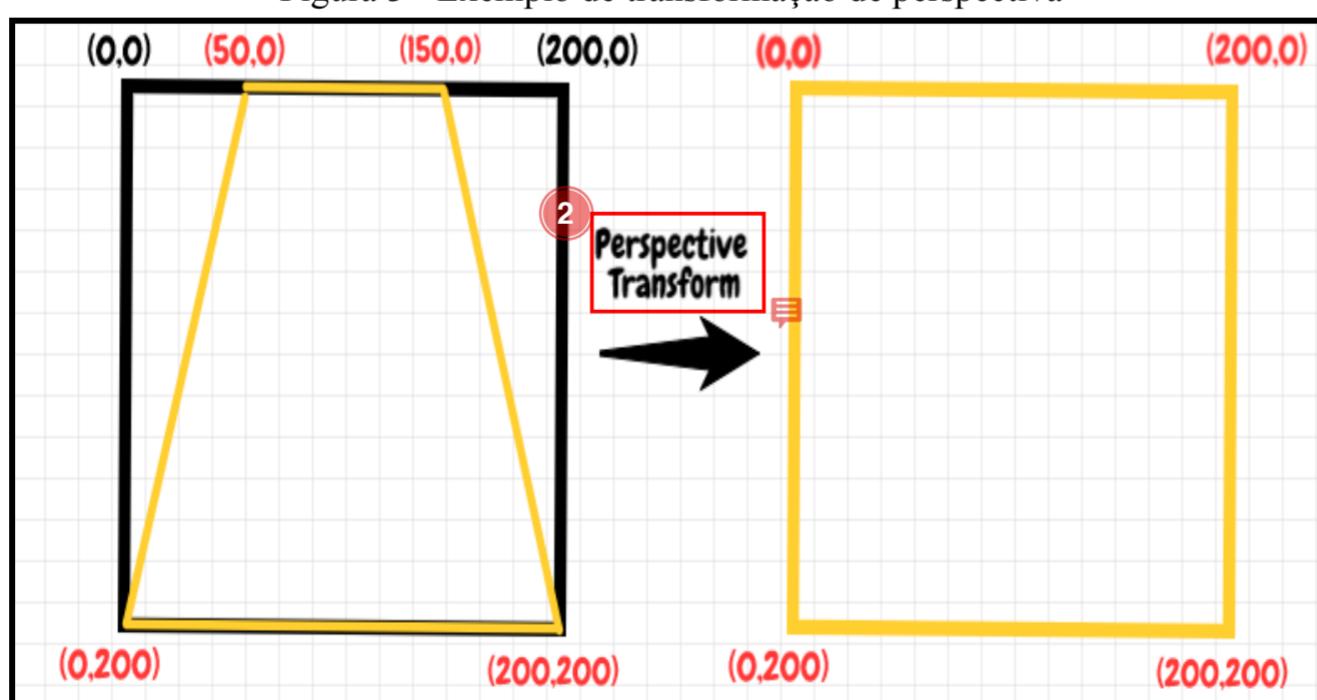
OpenCV

A transformação de perspectiva é usada para mudar a perspectiva de um objeto, por exemplo, quando os olhos humanos veem um objeto de perto, eles parecem serem maiores em relação aos objetos mais longes, isso é chamado de perspectiva. No geral, a transformação da perspectiva trata da conversão do mundo 3D em imagem 2D. O mesmo princípio sobre o qual a visão humana e câmera funcionam (TUTORIALSPPOINT, 2013).

Em transformação de perspectiva precisamos fornecer os pontos da imagem a partir dos quais queremos coletar informações, alterando a perspectiva. Também precisamos fornecer os pontos dentro dos quais queremos exibir nossa imagem. Em seguida, obtemos a transformação de perspectiva dos dois conjuntos de pontos fornecidos e a envolvemos com a imagem original (GEEKSFORGEEKS, 2020).

A Figura 3 apresenta um exemplo utilizando a transformação de perspectiva, que tem os pontos iniciais $[[50,0], [150,0], [0,200], [200,200]]$ e deve ser transformada para os novos pontos $[[0,0], [200,0], [0,200], [200,200]]$.

Figura 3 - Exemplo de transformação de perspectiva



Fonte: Shaikh (2020).

Traduzir.

2.4 FILTRO DE CANNY

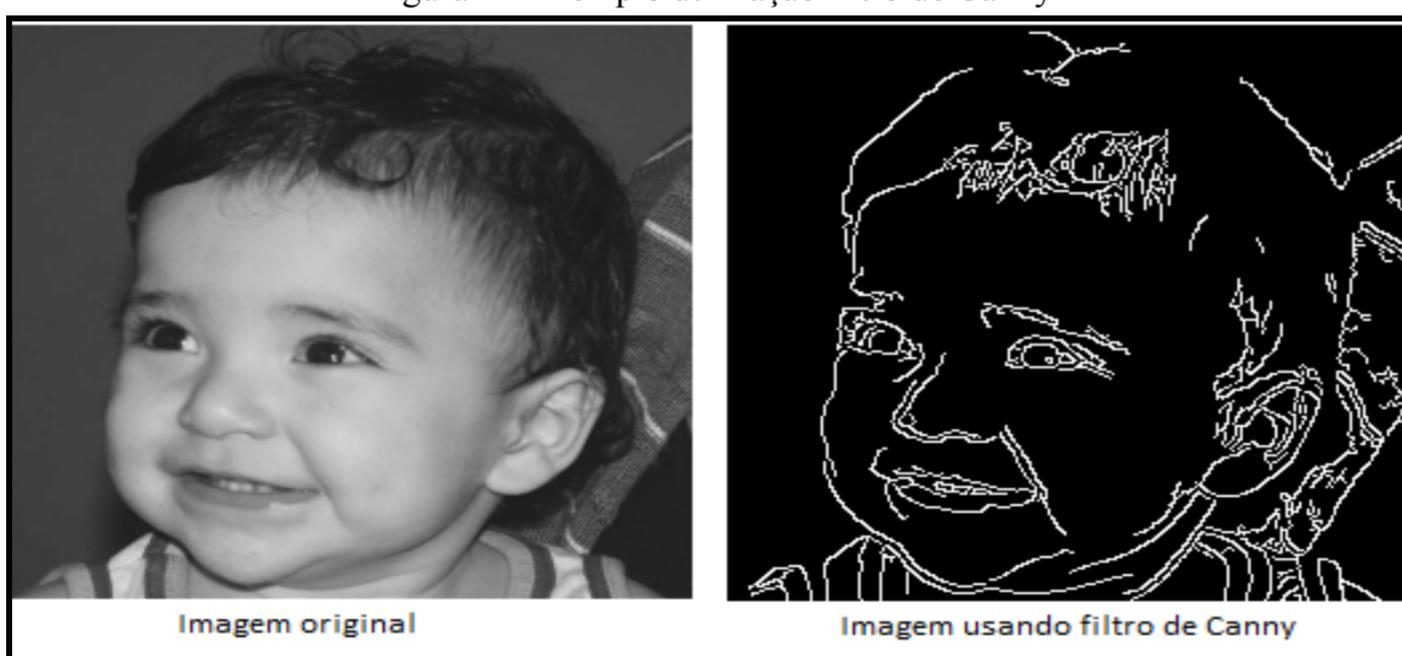
O OpenCV disponibiliza também o algoritmo de Canny. Esse algoritmo foi desenvolvido em 1986 por John F. Canny. O detector de bordas de Canny é reconhecido como um dos mais rápidos e eficientes algoritmos para encontrar descontinuidade em uma imagem. A partir de um conjunto de parâmetros, ele produz como resultado uma imagem binária contendo as bordas obtidas a partir de uma imagem (BRITO, 2015).

O primeiro passo do algoritmo de filtro de Canny é remover o ruído da imagem com um filtro gaussiano, isso se faz necessário devido a detecção de bordas ser suscetível a ruídos na imagem. Após isso a imagem é filtrada com um kernel Sobel na direção horizontal e vertical para obter a primeira derivada na direção horizontal e na direção vertical. A partir dessas duas imagens se pode encontrar o gradiente de borda e a direção de cada *pixel* (BRITO, 2015).

Em seguida uma varredura completa da imagem é feita para remover quaisquer *pixels* indesejados que possam não constituir a borda. Para isso, a cada *pixel*, é verificado se ele é um máximo local em sua vizinhança na direção do gradiente, se o valor da magnitude do gradiente for inferior a pelo menos um de seus vizinhos, é colocado 0, caso contrário é considerado para próxima etapa (BRITO, 2015).

O último passo, a Limiarização com histerese é usada para a quebra do contorno, ou seja, decidir quais arestas são realmente arestas e quais não são. Para isso, precisa-se de dois valores de limite, *minVal* e *maxVal*. Quaisquer arestas com gradiente de intensidade maior que *maxVal* são consideradas arestas e aquelas abaixo de *minVal* são consideradas não arestas, portanto, são descartadas. Aqueles que estão entre esses dois limites são classificados como bordas ou não com base em sua conectividade. Se eles estiverem conectados a pixels de "aresta segura", eles serão considerados parte das arestas. Caso contrário, eles também são descartados (BRITO, 2015). A Figura 4 mostra um exemplo da utilização do filtro de Canny.

Figura 4 - Exemplo utilização filtro de Canny



Fonte: Brito (2015).

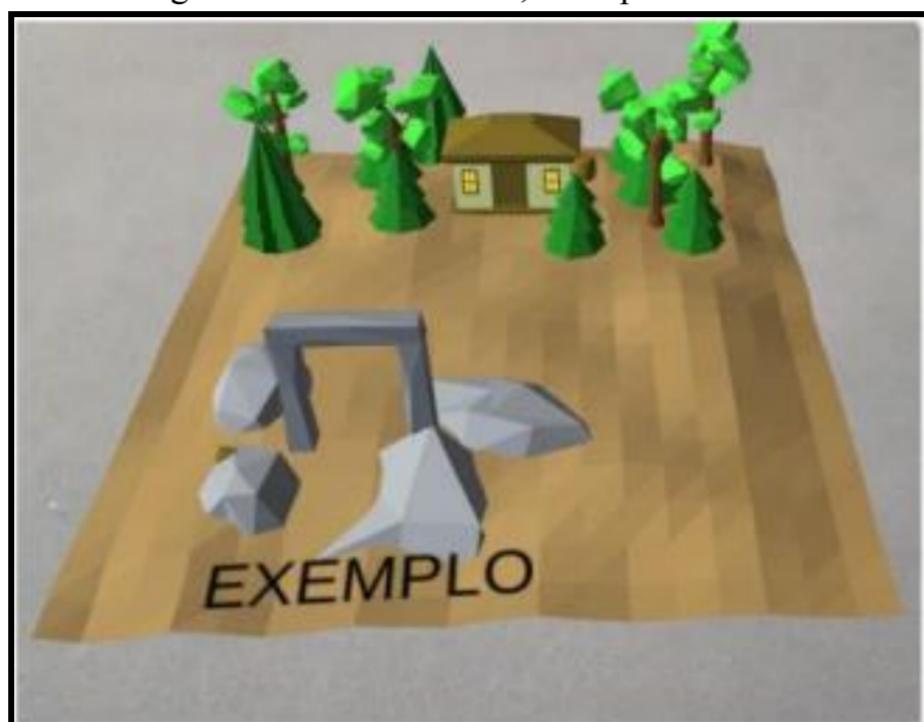
2.5 TRABALHOS CORRELATOS

Foram selecionados três trabalhos correlatos que apresentam semelhanças com o proposto neste trabalho. Na seção 2.1 é apresentado o iAR ferramenta desenvolvida por Hess (2011), capaz executar os recursos da realidade aumentada na plataforma iOS. Na seção 2.2 é apresentada a ferramenta animar, desenvolvida por Reiter (2018) que tem como objetivo criar cenas animadas utilizando os recursos da realidade aumentada. Por fim, na seção 2.3 é apresentado um protótipo desenvolvido por Kumagai (2011), utilizando técnicas de reconhecimento de objetos sem marcadores.

Quadro 1 - iAR

Referência	Hess (2011)
Objetivos	Executar os recursos da realidade aumentada na plataforma iOS, que disponibilize opções para a interação do usuário com os objetos virtuais na tela.
Principais funcionalidades	iAR foi desenvolvida em formato Application Programming Interface (API), com funções para facilitar a reutilização da ferramenta para qualquer objetivo.
Ferramentas de desenvolvimento	A ferramenta foi desenvolvida em C++ e explora o uso de algumas ferramentas como OpenGL ES versão 2.0 para apresentação de objetos 3D, bibliotecas como ArUco para identificar a posição e a orientação dos objetos 3D a partir de marcadores, OpenCV biblioteca para tratamentos de imagens digitais e Libobj um analisador de arquivos com extensão .obj, que auxilia na criação de objetos 3D a partir de definições geométricas contidas no arquivo.
Resultados e conclusões	Segundo Hess (2011, p. 58), foi identificado problemas na geração de matrizes de visualização e de projeção inconsistente devido ao tamanho e orientação das imagens capturadas, sendo

Figura 6 - Marcador cena, visto pela câmera



Fonte: Reiter (2019, p. 63).

Quadro 3 - Estudo e implementação de técnicas de realidade aumentada

Referência	Kumagai (2011)
Objetivos	Possibilitar a utilização dos recursos da realidade aumentada sem marcadores.
Principais funcionalidades	Permite a inserção dos elementos virtuais no ambiente real, utilizando informações naturalmente presentes como arestas, texturas ou a própria estrutura da cena sem a necessidade de marcadores.
Ferramentas de desenvolvimento	Para implementação do projeto Kumagai (2011, p. 33) utilizou a API OpenCV com a ferramenta Microsoft Visual Studio 2010 com a linguagem C++. Já a biblioteca OpenGL foi utilizada para desenhar os objetos 3D na tela.
Resultados e conclusões	A técnica de reconhecimento de objetos foi implementada com sucesso e mostrou-se robusta a movimentos suaves e variação de luminosidade mesmo com classificador de qualidade baixa. Porém, em algumas cenas o classificador identifica áreas de interesse mesmo quando a mesma não é de interesse. O protótipo implementado possui algumas limitações, como, de não projetar mais de um objeto virtual na mesma cena, e pela questão de variação de luminosidade e movimentos bruscos, sendo necessário equipamentos de alta qualidade. (Kumagai, 2011, p. 42).

Fonte: elaborado pelo autor.

1 Remover ponto final.

3 DESCRIÇÃO DO APLICATIVO

Este capítulo está organizado em 5 seções. A seção 3.1 apresenta a especificação do aplicativo, utilizando o diagrama de casos de uso e os requisitos. Na seção 3.2 é apresentado os diagramas de classes do aplicativo. Na seção 3.3 apresenta a arquitetura utilizada no aplicativo. Na seção 3.4 apresenta uma visão geral do aplicativo, bem como seu objetivo, funcionamento e forma de utilização do aplicativo por parte do usuário. E por fim, na seção 3.5 é destacado as principais técnicas implementadas para a construção das funcionalidades.

3.1 ESPECIFICAÇÃO

Esta seção apresenta os Requisitos Funcionais (RF) e Não Funcionais (RNF) das aplicações. Inicialmente o Quadro 4 apresenta os requisitos funcionais, bem como a rastreabilidade com o diagrama de caso de uso da Figura 7. O Quadro 5 apresenta os requisitos não funcionais.

Quadro 4 - Requisitos Funcionais

Requisitos Funcionais	Caso de uso
RF01: permitir selecionar um objeto pré-definido no aplicativo	UC-01
RF02: permitir selecionar um arquivo localizado no aplicativo	UC-02
RF03: permitir selecionar um objeto já desenhado anterior	UC-03
RF04: permitir rotacionar um objeto na tela do dispositivo	UC-04
RF05: permitir gerar cena	UC-05
RF06: gerar as bordas do objeto selecionado	UC-06
RF07: sobrepor na tela do dispositivo as bordas do objeto selecionado	UC-07
RF08: comparar desenho do usuário com bordas do objeto selecionado	UC-08

Fonte: elaborado pelo autor.

2 os requisitos e diagrama de casos de uso

3 e Requisitos Não

4 RFs

5 RNFs.

Colocar o ponto final.

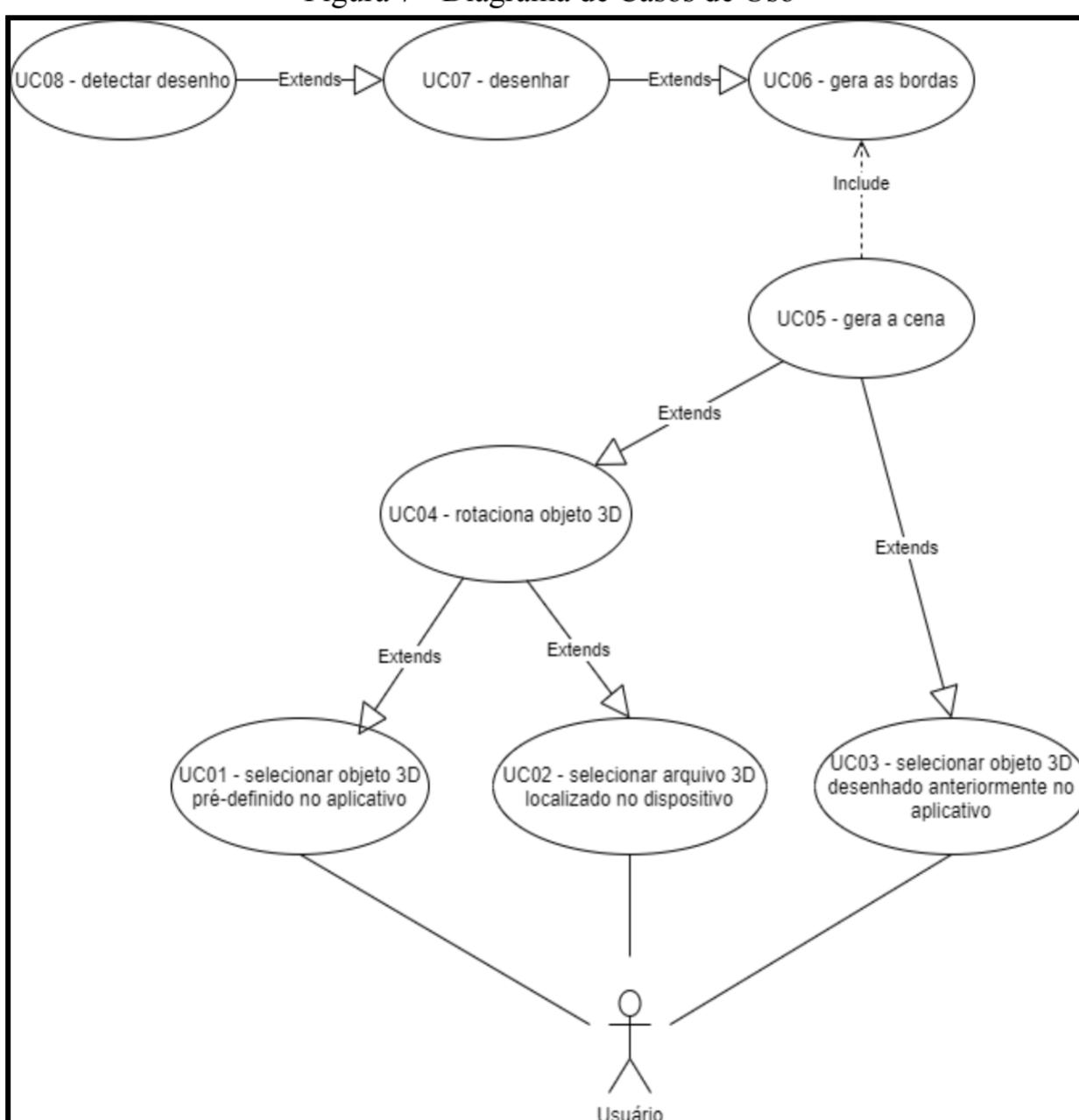
Quadro 5 - Requisitos Não Funcionais

Requisitos Não Funcionais
RNF01: ser desenvolvida para iOS e Android
RNF02: utilizar o ambiente Unity para desenvolvimento
RNF03: utilizar as bibliotecas ArUco para detectar marcadores
RNF04: utilizar biblioteca OpenCV para Unity para reconhecimento de imagens
RNF05: utilizar recurso Unity para salvar informações no dispositivo

Fonte: elaborado pelo autor.

Para representar as principais funcionalidades do módulo administrativo, criou-se um diagrama de caso de uso conforme apresentado na Figura 7.

Figura 7 - Diagrama de Casos de Uso



Fonte: elaborado pelo autor.

Foi identificado um único ator, que é o usuário, aquele que irá utilizar o aplicativo. O caso de uso UC01 – selecionar objeto 3D pré-definido no aplicativo permite ao usuário selecionar um objeto cadastrado no aplicativo durante o desenvolvimento. O caso de uso UC02 – selecionar arquivo 3D localizado no dispositivo permite ao usuário criar ou baixar da internet seu próprio arquivo 3D e importar para o dispositivo. O caso de uso UC03 – selecionar objeto 3D desenhado anteriormente no aplicativo permite ao usuário selecionar um objeto que já tenha desenhado em outro momento no aplicativo e que deseja desenhá-lo novamente.

O caso de uso UC04 – rotaciona objeto 3D permite ao usuário rotacionar o objeto na posição que desejar. O caso de uso UC06 – gera as bordas descreve a funcionalidade que gera uma imagem com as bordas do objeto selecionado pelo usuário. Esse caso de uso será executado após o caso de uso UC05 – gera a cena for executado pelo usuário. O caso de uso UC07 – desenhar, permite ao usuário criar seu desenho tendo como base as bordas do objeto selecionado sobreposto a tela do dispositivo. E, por fim, o caso de uso UC08 – detectar desenho descreve a funcionalidade que compara o desenho feito pelo usuário com as bordas do objeto selecionado.

3.2 DIAGRAMA DE CLASSES

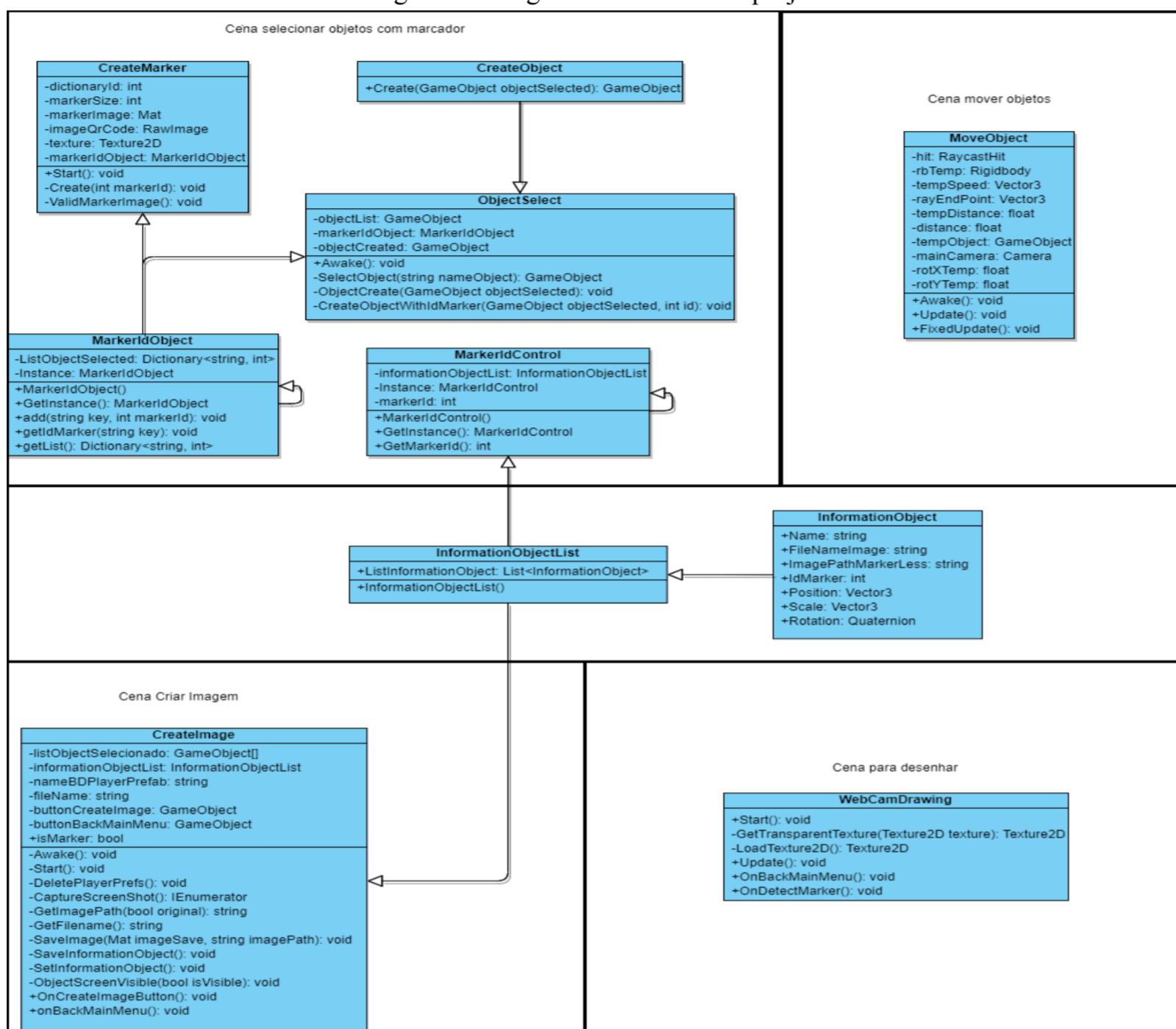
A estrutura de classes criada para o aplicativo está exibida na Figura 8. Nela é apresentado um diagrama de classes que mostra uma visão de como elas estão estruturadas e relacionadas no aplicativo. O diagrama foi separado por

Fonte courier.

criada

cenas, contendo as cenas de selecionar objeto com marcadores, cena de mover objetos, cena de criar Imagem e a cena para desenhar.

Figura 8 - Diagrama de classes do projeto



Fonte: elaborado pelo autor.

Na cena selecionar objetos com marcadores **2** tem a classe CreateMarker que é responsável pela estrutura de criação de marcadores ArUco com seu respectivo id de forma dinâmica. A classe MarkerIdObject é responsável de armazenar em memória todos os objetos 3D e seus **ids**. A classe CreateObject é uma classe genérica para criar um objeto e adicionar a cena. A classe ObjectSelect é responsável por selecionar o objeto 3D e vincular o **id** ArUco a ele. A classe MarkerIdControl é responsável pelo controle da criação de **ids** ArUco não permitindo que mais de um objeto tenha o mesmo **id**.

3 Temos as classes **InformationObject** e **InformationObjectList** que não estão ligadas a uma cena e sim ao projeto como um todo. A classe **InformationObject** é responsável pelas informações do objeto 3D, como nome, nome da imagem, caminho da imagem, id ArUco, a posição, rotação e escala do objeto. A classe **InformationObjectList** é responsável por criar uma lista da classe **InformationObject**. Essas classes são do tipo **Serializable** que permite elas serem convertidas para tipo **Json** e serem salvas na memória do dispositivo, como mostra o exemplo no Quadro 6.

Quadro 6 - Exemplo conversão para Json

```
145     string informationList = JsonUtility.ToJson(informationObjectList);
146     PlayerPrefs.SetString(nameBDPlayerPrefab, informationList);
```

Fonte: elaborado pelo autor.

A cena mover objeto é composto por apenas uma classe, a **MoveObject** é responsável por identificar o objeto que está sendo clicado na cena e implementar seu movimento pela tela do dispositivo. Temos a cena criar imagem com a classe **CreateImage** que é responsável por tirar um print da tela com o objeto selecionado na posição desejada e salvar no dispositivo, guardar informações do objeto como nome, posição, rotação e caminho da imagem na

cenas (seguindo o padrão definido pelo Unit), contendo

Fonte courier.

As

InformationObjectList
não

MoveObject, que é

por capturar uma
imagem da tela

rotação, caminho

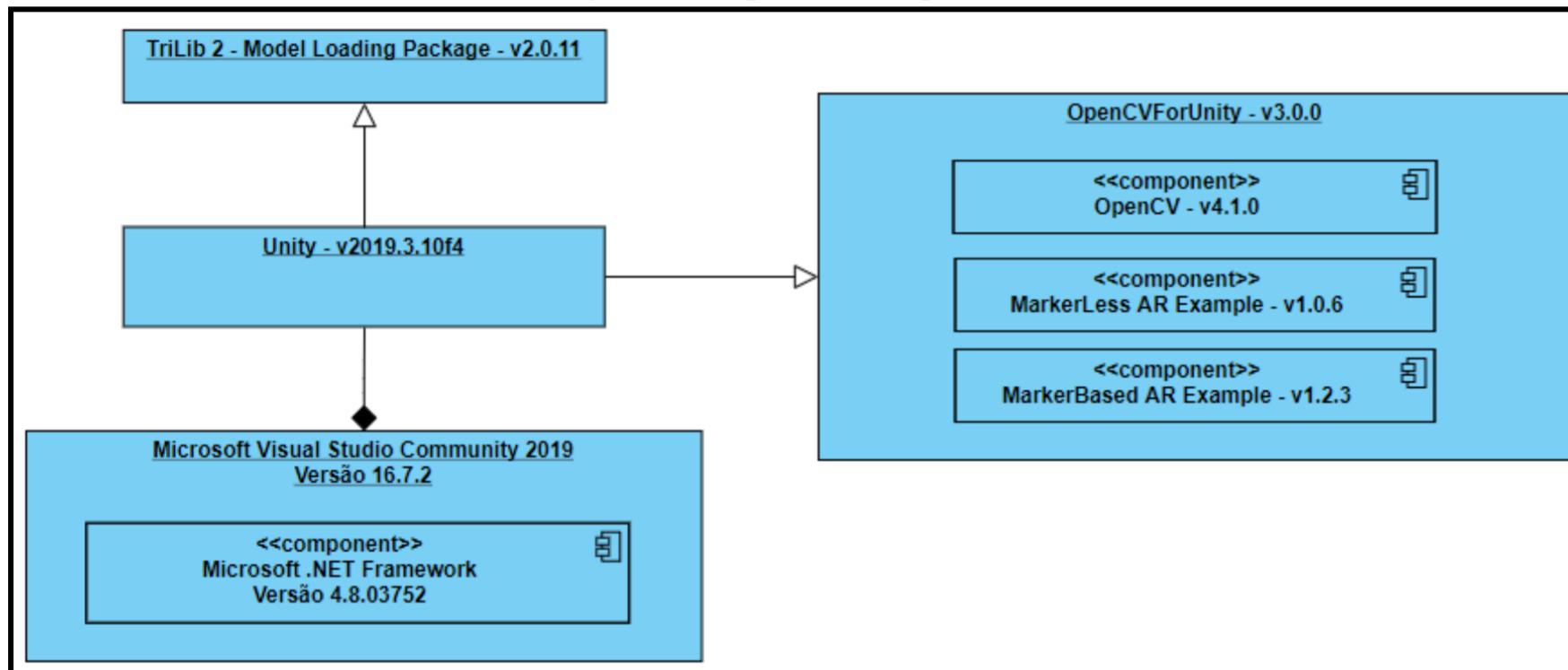
memória do dispositivo e criar as bordas do objeto selecionado. E por fim a classe `WebCamDrawing` presente na cena para desenhar é responsável em apresentar as bordas sobreposta na tela dispositivo, permitindo ao usuário criar seu desenho tendo como base o objeto 3D selecionado por ele.

3.3 ARQUITETURA

A arquitetura foi montada conforme a representação no diagrama de arquitetura na Figura 9. Foi utilizado a plataforma Unity 3D na versão 2019.3.10f1 com scripts na linguagem C# juntamente com a plataforma Microsoft Visual Studio Community 2019 versão 16.7.2 para controle dos componentes das cenas e OpenCV versão 4.1.0 para realização do processamento de imagem. Como OpenCV não integração direta com a Unity foi utilizado o asset OpenCVForUnity na versão 3.0.0, como descrito na seção 2.1.

Foi utilizado o asset Markerless AR Example como base para o desenvolvimento da realidade aumentada sem marcadores pré-definidos e o asset MarkerBased AR Example como base para o desenvolvimento da realidade aumentada com marcadores ArUco, ambos os assets estão disponíveis na Unity asset Store para download gratuito, porém tem como requisito para funcionamento o asset do OpenCVForUnity que é pago, como descrito na seção 2.1. Outro asset utilizado nesse projeto foi o TriLib 2 - Model Loading Package que permite importar modelos 3D para uma cena do Unity em tempo de execução. O Quadro 7 apresenta o link de cada asset utilizado nesse projeto.

Figura 9 – Arquitetura do aplicativo



Fonte: elaborado pelo autor.

Quadro 7 - Links da loja

Asset	Link da loja
OpenCVForUnity	https://assetstore.unity.com/packages/tools/integration/opencv-for-unity-21088
Markerless AR Example	https://assetstore.unity.com/packages/templates/tutorials/markerless-ar-example-77560
MarkerBased AR Example	https://assetstore.unity.com/packages/templates/tutorials/markerbased-ar-example-29678?aid=101114ehR&utm_source=aff
TriLib 2 - Model Loading Package	https://assetstore.unity.com/packages/tools/modeling/trilib-2-model-loading-package-157548#reviews

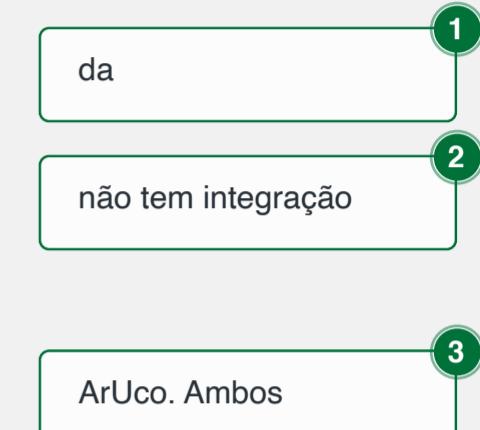
Fonte: elaborado pelo autor

3.4 VISÃO GERAL DO APLICATIVO

O aplicativo tem como principal objetivo permitir que qualquer tipo de usuário possa criar sua própria cena 3D com realidade aumentada. Ele permite ao usuário importar para o aplicativo um arquivo 3D baixado da internet ou criado por ele mesmo. A partir disso o aplicativo converte esse arquivo para um objeto 3D do Unity sendo possível selecionar, definir sua posição e criar uma cena 3D com ele para posteriormente ser desenhado pelo usuário e criado uma sobreposição no dispositivo com realidade aumentada. A Figura 10 detalha a sequência das atividades do aplicativo.

<https://assetstore.unity.com/packages/templates/tutorials/markerbased-ar-example-29678>

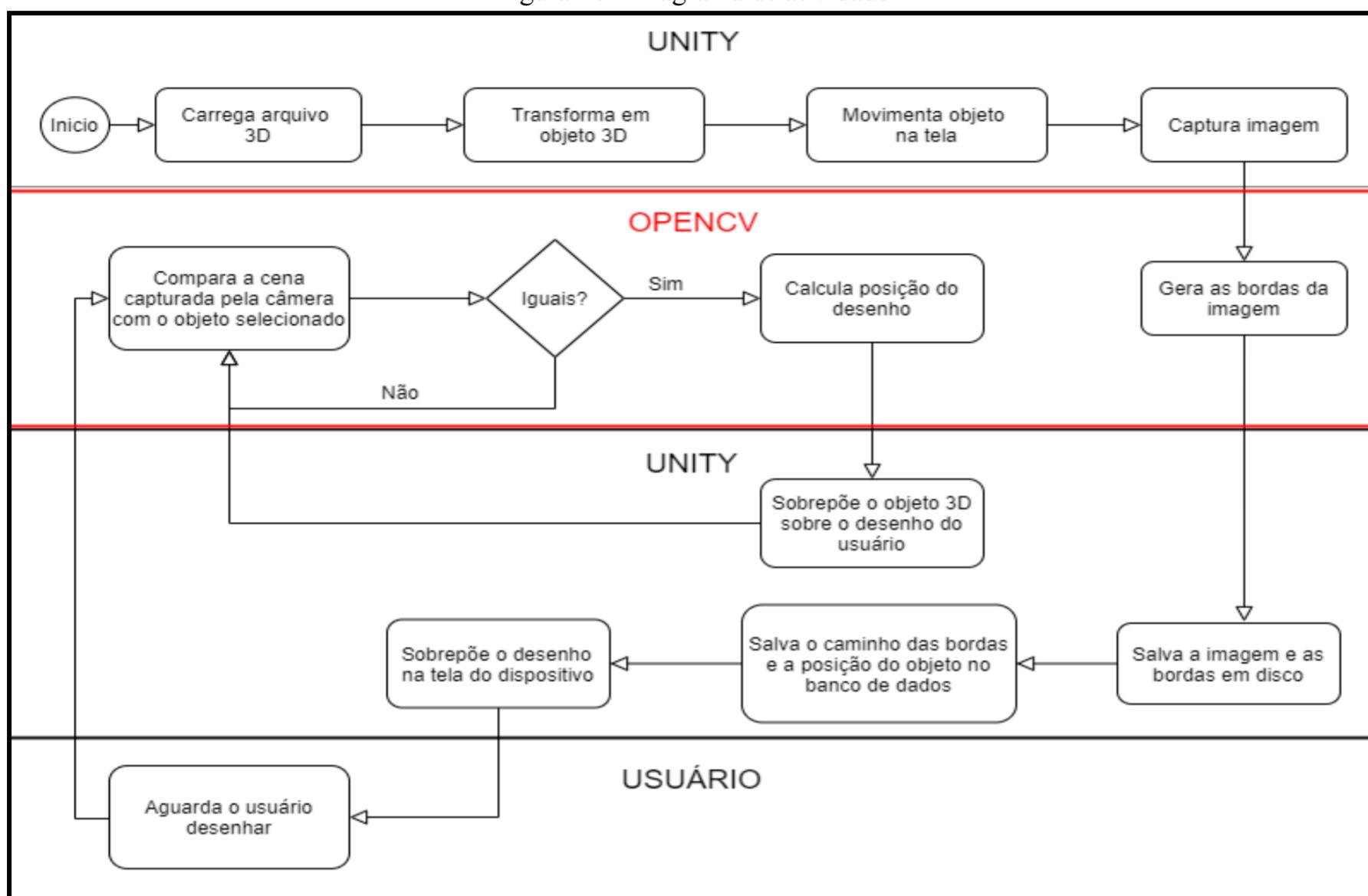
<https://assetstore.unity.com/packages/tools/modeling/trilib-2-model-loading-package-157548>



Melhor
<https://assetstore.unity.com/packages/templates/tutorials/markerbased-ar-example-29678>

Melhor
<https://assetstore.unity.com/packages/tools/modeling/trilib-2-model-loading-package-157548>

Figura 10 - Diagrama de atividade

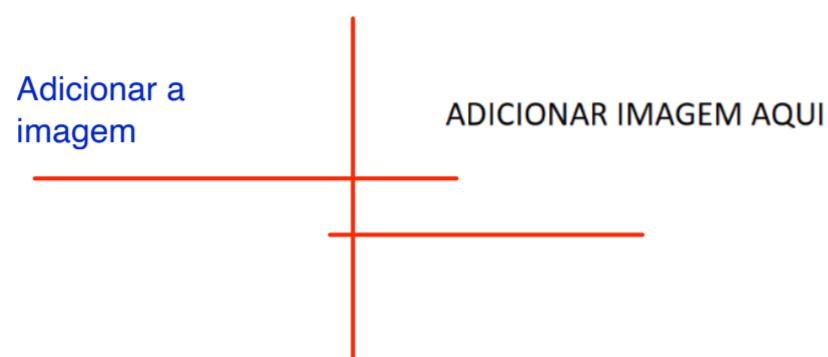


Fonte: elaborado pelo autor.

Ao iniciar o aplicativo será apresentado o menu principal, ele é composto pela opção Importar Objetos, que permite ao usuário importar para o aplicativo um arquivo 3D baixado da internet ou criado por ele mesmo, com as extensões (.fbx, .blender, .obj). Existe a opção Criar cena sem marcador, que permite o usuário criar uma cena sem a utilização de um marcador pré-definido, utilizando uma cena do mundo real através da câmera como marcador. Existe a opção Criar cena com marcador, que permite o usuário criar uma cena com um marcador ArUco pré-definido. Outra opção no menu é a Realidade Aumentada, essa opção abre a câmera do dispositivo e detecta marcadores ArUco e cenas do mundo real e compara com objetos criados no aplicativo. E caso a detecção retorne com sucesso, um objeto 3D vinculado ao marcador é sobreposto na tela do dispositivo. Por fim, o menu Desenhar novamente, que permite o usuário desenhar uma cena já criada nos menus anteriores. A Figura 11 detalha a tela do menu principal.

.blender e .obj

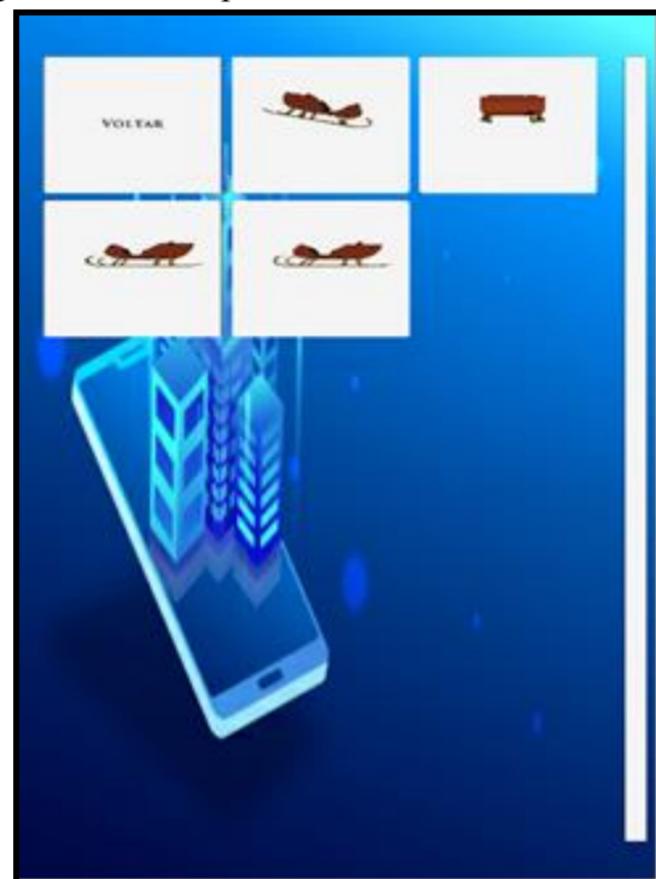
Figura 17 - Exemplo do aplicativo sobrepondo um objeto



Fonte: elaborado pelo autor.

Na última opção do menu principal **Desenhar novamente**, traz uma lista de objetos já criados anteriormente através dos menus **Gerar cena sem marcador** e **Gerar cena com marcador**. Ao ser clicado em um dos itens da lista, o usuário será redirecionado para tela de desenho como na Figura 14, sendo possível fazer um novo desenho para ser detectado. A Figura 18 mostra a tela de seleção de objetos já criados.

Figura 18 - Exemplo de a tela desenhar novamente



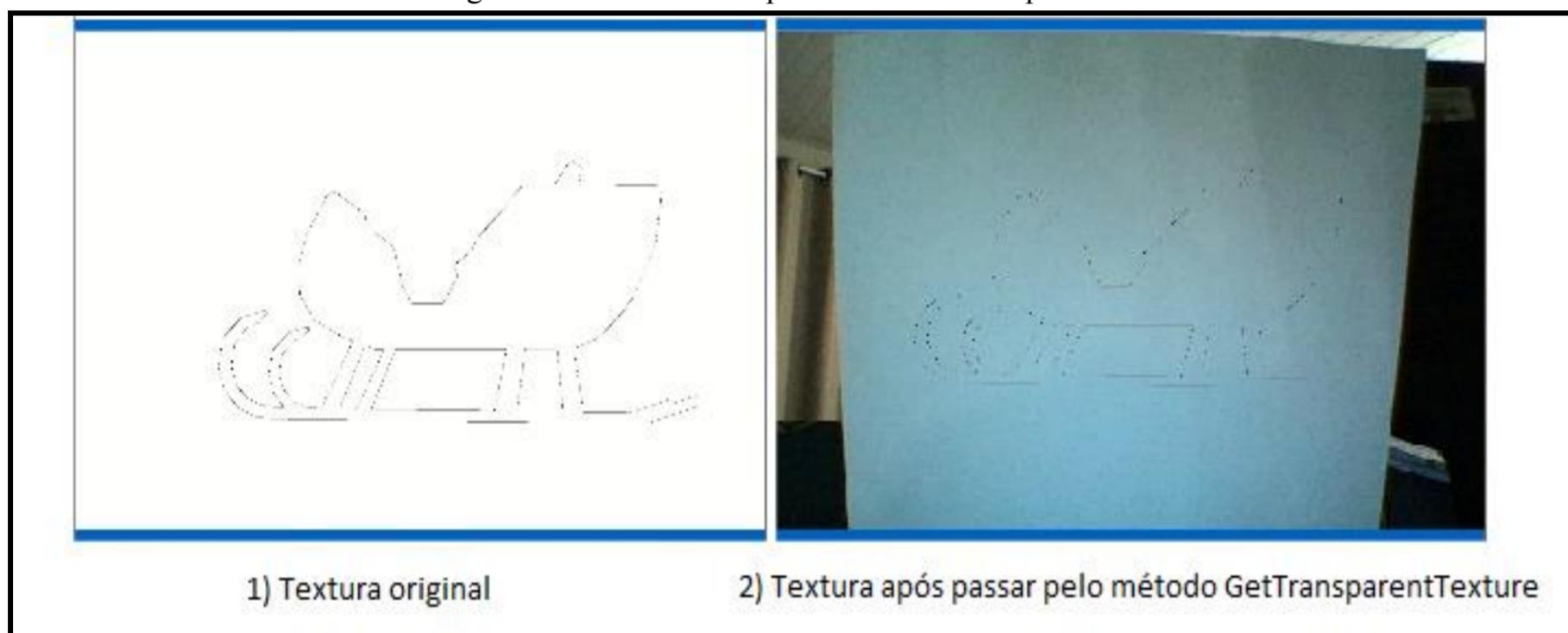
Fonte: elaborado pelo autor.

3.5 IMPLEMENTAÇÃO

Para captura das imagens através da câmera foi utilizado o método `GetMat` através do componente `WebCamTextureToMatHelper` que retorna um objeto do tipo `Mat` (específico do OpenCV) contendo a imagem da cena. Em seguida é utilizado o método `fastMatToTexture2D` do componente `Utils` que converte um objeto do tipo `Mat` para o tipo `Texture2D` para ser utilizado no Unity.

Para geração das bordas foi utilizado o algoritmo filtro de Canny (ver seção 2.4). O OpenCV possui o método `Canny` do componente `Imgproc` que aplica o algoritmo filtro de Canny em uma imagem do tipo `Mat`. E assim retorna uma imagem com o fundo preto e as bordas da imagem branca, sendo necessários inverter as cores deixando o fundo branco e as bordas pretas. Para isso foi utilizado o método `bitwise_not` que recebe uma imagem do tipo `Mat` e inverte as cores. A Figura 19 mostra o processo para gerar as bordas, sendo a primeira imagem a original, a segunda após o filtro de Canny e a terceira após utilizar o método `bitwise_not` para inverter as cores.

Figura 20 – Bordas sobrepostas na tela do dispositivo



Fonte: elaborado pelo autor.

Para comparação do desenho feito pelo usuário com o objeto selecionado, foi utilizado o método `findHomography` do componente `Calib3d` do OpenCV. O Quadro 9 detalha a usabilidade do método que recebe como primeiro e segundo parâmetro um conjunto de pontos da imagem original e da imagem que está sendo detectada. O terceiro parâmetro é o método usado para calcular a homografia, que nesse aplicativo foi utilizado o método `RANSAC`. O quarto parâmetro define a quantidade de erro permitido para tratar os pontos, utilizamos quantidade máxima de 3. O sexto parâmetro define o número de iterações, nesse aplicativo foi utilizado 2000 iterações. E por último é definido o nível de confiança entre 0 e 1, que nesse aplicativo foi utilizado 0.5. O método `findHomography` irá encontrar e retornar a transformação de perspectiva entre os pontos chave correspondentes da imagem original e a que está sendo detectada.

Quadro 9 - Método FindHomography

```

453
454 =
455
456
457
458
459
460
    Calib3d.findHomography (
        srcPoints,
        dstPoints,
        Calib3d.FM_RANSAC,
        3,
        inliersMask, 2000, 0.5);

```

Fonte: elaborado pelo autor.

Citar quadro no texto. 1
Para calcular a posição que o objeto está, foi utilizado o método `perspectiveTransform` do componente `Core` do OpenCV, que recebe como primeiro parâmetro uma matriz de pontos flutuantes do objeto original. O segundo parâmetro é uma matriz de pontos flutuantes que irá receber o resultado da transformação. Essa matriz deve ter o mesmo tamanho do elemento original. E por fim, recebe como último parâmetro a matriz de homografia através do método `findHomography`.

Quadro 10 - Método perspectiveTransforme

```

281
282     Core.perspectiveTransform (m_pattern.points2d, info.points2d, info.homography);
283

```

Fonte: elaborado pelo autor.

4 RESULTADOS

A seção de resultados foi dividida em três partes. Na primeira são apresentados os testes feitos em relação aos *assets* utilizados. Na segunda são descritos os testes de funcionalidades. E por fim são apresentadas as abordagens feitas na usabilidade do aplicativo.

4.1 TESTES DOS ASSETS

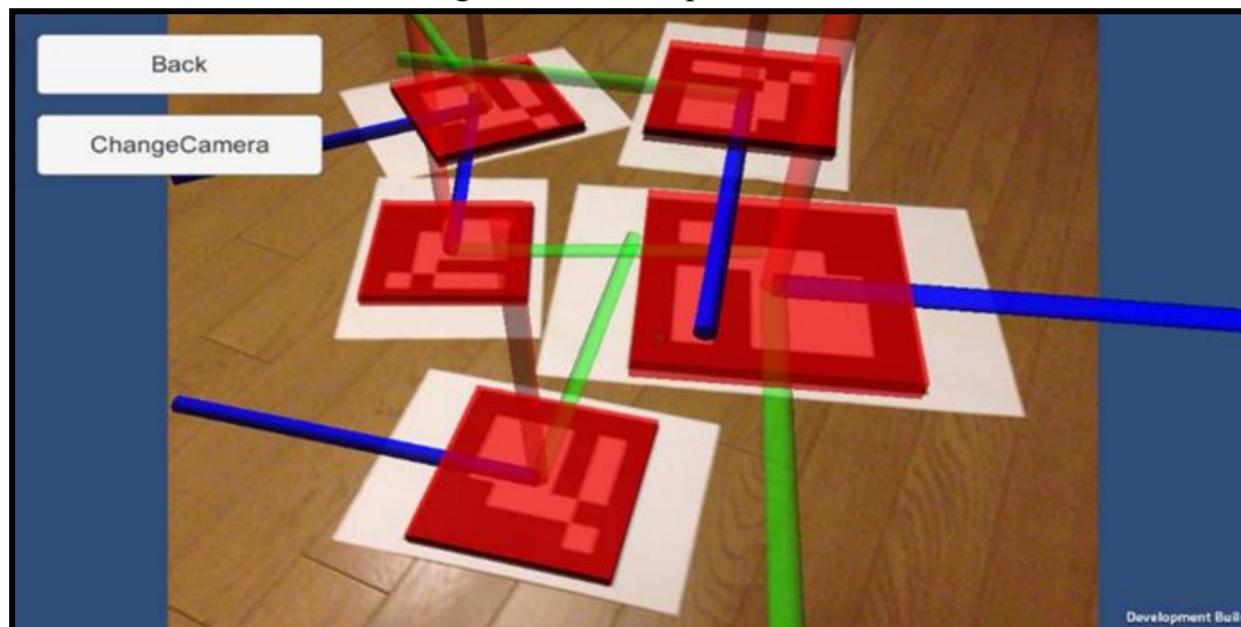
O primeiro passo na construção do aplicativo, foi realizar testes no *asset* do OpenCVForUnity. O primeiro teste foi construir uma aplicação utilizando a biblioteca ArUco, permitindo que o usuário criasse várias cenas e o aplicativo

aplicativo foi

fosse vinculando de forma automática marcadores para essas cenas, desse modo quando o usuário apontasse a câmera para um marcador o aplicativo se encarregava de identificar o marcador e a cena vinculada a ele para em seguida sobrepor na tela do dispositivo.

O desenvolvimento teve bastante êxito, tendo em vista que a biblioteca ArUco é muito eficiente na detecção de seus marcadores

Figura 21 - Exemplo ArUco

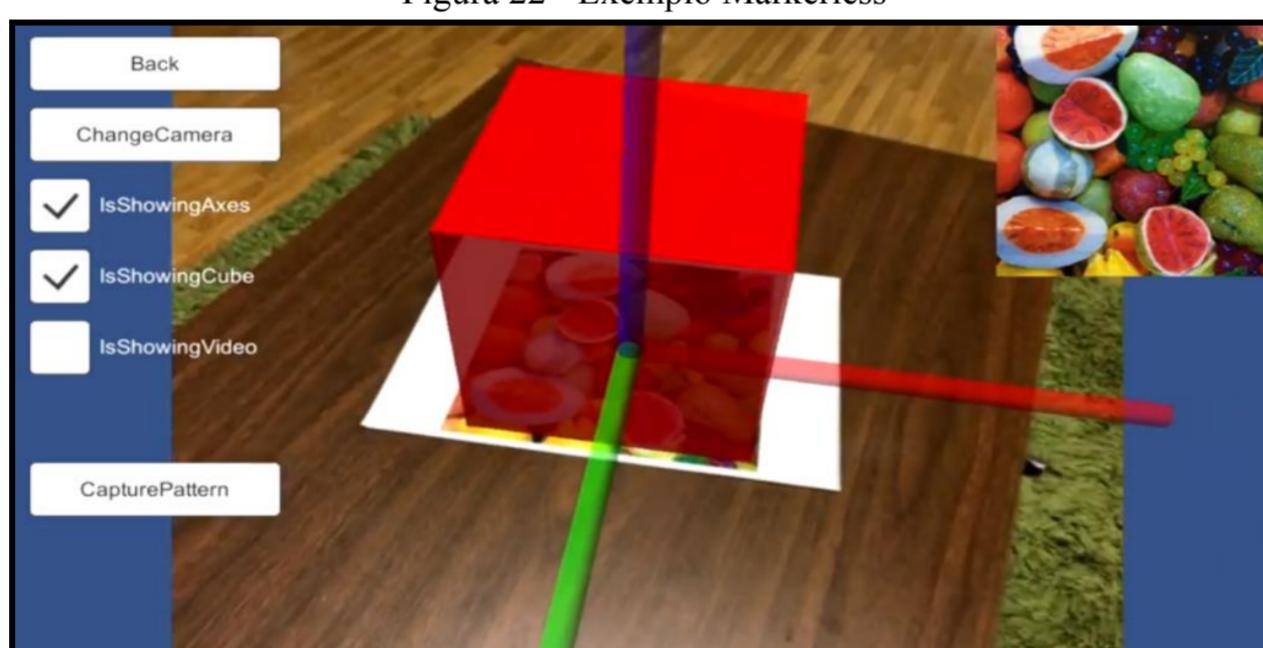


Fonte: elaborado pelo autor.

Outro *asset* utilizado nos testes foi o MarkerLess AR Example que utiliza cenas do mundo real como marcador para sobrepor objetos 3D na tela do aplicativo. O desenvolvimento foi parecido com o *asset* do ArUco, o aplicativo permite que o usuário selecione um objeto 3D do aplicativo e movimente-o na posição desejada, o aplicativo irá tirar uma foto desse objeto e fazer dele um marcador, assim quando o usuário apontar a câmera do dispositivo para a mesma imagem o aplicativo irá comparar e detectar a semelhança.

Os testes feitos com esse *asset* tiveram bastante êxito, utilizando as imagens disponibilizado pelo *asset* e também com outras imagens, se mostrou muito eficaz na comparação detecção das imagens.

Figura 22 - Exemplo Markerless



Fonte: elaborado pelo autor.

4.2 TESTE DE FUNCIONALIDADE

Durante o desenvolvimento do aplicativo foram realizados testes constantes para garantir que as funcionalidades estivessem de acordo com o esperado e verificar possíveis erros e identificar dificuldades que os usuários poderiam enfrentar durante a utilização da aplicação, a fim de sanar possíveis falhas.

O desenvolvimento do jogo foi realizado de forma iterativa, com constante refatoração de código para atender a novos requisitos do jogo. As funcionalidades foram construídas para atender a objetivos específicos e foram refatoradas para atender objetivos de forma genérica conforme a necessidade. Assim, testes foram realizados constantemente para garantir que as funcionalidades continuavam de acordo em todas as partes do aplicativo.

Foram realizados testes tanto no emulador do Unity quanto em um dispositivo real. Os testes levaram em considerações a forma como os usuários poderiam utilizar o aplicativo. Foram observados diversos aspectos, como a

cenas. Desse

ArUco, onde o

desejada. O aplicativo

marcador. Assim

esperado. E também
verificar

Hum, não
desenvolvesse um
jogo???

mover na posição desejada e criar seu próprio desenho com base nas bordas sobreposta na tela do dispositivo. As bordas são criadas pelo aplicativo a partir do objeto selecionado pelo usuário, após o desenho ser finalizado, o aplicativo analisa o desenho e caso haja certa semelhança com as bordas do objeto 3D selecionado anteriormente, esse objeto 3D é sobreposto na tela do dispositivo, porém, devido a limitação da biblioteca OpenCV na comparação das bordas com o desenho criado pelo usuário, em alguns momentos o objeto 3D fica “pulando” pela tela do dispositivo.

A ferramenta Unity3D se mostrou eficiente no desenvolvimento do aplicativo e de fácil aprendizado, uma vez que boa parte das funcionalidades utilizadas foram aprendidas no decorrer do desenvolvimento do projeto. A biblioteca OpenCV integrado no asset OpenCVForUnity foi muito importante para o desenvolvimento desse trabalho e se mostrou eficaz e de fácil utilização para o propósito desse trabalho, apesar de apresentar limitação na comparação das bordas.

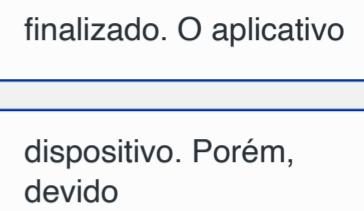
A biblioteca ArUco integrado no asset do OpenCVForUnity também se mostrou muito eficiente para o desenvolvimento desse trabalho, pois sua técnica de detecção de marcadores é muito rápida e eficaz, dando uma experiência muito boa para o usuário. Além disso sua documentação é muito robusta trazendo conforto no desenvolvimento da aplicação.

Embora o aplicativo tenha cumprido seus objetivos, foi identificado possibilidade para continuação e extensão da pesquisa deste projeto.

- a) estudo para melhorar a detecção das bordas;
- b) alterar as cores do objeto 3D conforme o usuário pinte na folha de papel;
- c) melhorar o desempenho da câmera quando tiver muitos objetos criados;
- d) permitir adicionar mais de um objeto 3D na cena;
- e) melhorar a interface mostrada para o usuário.

REFERÊNCIAS

- ARUCO. **Detection of ArUco Markers.** [S.I.], [2020]. Disponível em: <https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html>. Acesso em: 14 set. 2020.
- AVA. **Aplicaciones de la Vision Artificial.** [S.I.], [2018?]. Disponível em: <<http://www.uco.es/investiga/grupos/ava/node/26>>. Acesso em: 14 set. 2020.
- BRITO, Agostinho. **8. Detecção de bordas com o algoritmo de Canny.** [S.I.], 2015. Disponível em: <https://agostinhobritojr.github.io/tutorial/pdi/#_detec%C3%A7%C3%A3o_de_bordas_com_o_algoritmo_de_canny>. Acesso em: 15 nov. 2020.
- COSTA, José W. A. et al. Realidade Virtual Aumentada Aplicada como Ferramenta de Apoio ao Ensino. **Revista Tecnologia em Projeção**, [Brasília], v.2, n.1, p. 11-15, jun. 2011.
- ENOX SOFTWARE. **OpenCV for Unity.** [S.I.], 2016. Disponível em: <<https://enoxsoftware.com/opencvforunity/>>. Acesso em: 14 set. 2020.
- GEEKSFORGEEKS. **Perspective Transformation – Python OpenCV.** [S.I.], 2020. Disponível em: <<https://www.geeksforgeeks.org/perspective-transformation-python-opencv/>>. Acesso em: 02 nov. 2020.
- HESS, Jonathan. **Explorando modelos virtuais 3D com realidade aumentada no SDK do Iphone.** 2011a. 76 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.
- KIRNER, Claudio; SISCOUTTO, Robson. **Realidade Virtual e Aumentada: Conceitos, Projeto e Aplicações.** Petrópolis, RJ: [s.n.], 2007.
- KUMAGAI, Anderson. **Realidade Aumentada Utilizando Marcadores Não Convencionais.** 2011. 47 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha de Marília, Marilia.
- OPENCV. **OpenCV, about.** [S.I.], [2019]. Disponível em: <<https://opencv.org/about/>>. Acesso em: 14 set. 2020.
- OPENCV. **Conceitos básicos da homografia explicados com código.** [S.I.], [2020]. Disponível em: <https://docs.opencv.org/master/d9/dab/tutorial_homography.html>. Acesso em: 16 nov 2020.
- OZLU, Ahmet. **Marker-less Augmented Reality by OpenCV and OpenGL.** [S.I.], [2018]. Disponível em: <<https://medium.com/@ahmetozlu93/marker-less-augmented-reality-by-opencv-and-opengl-531b2af0a130>>. Acesso em: 15 set. 2020.
- PERI, Abhninav. **Usando homografia para estimativa de pose em OpenCV.** [S.I.], 2017. Disponível em: <<https://medium.com/analytics-vidhya/using-homography-for-pose-estimation-in-opencv-a7215f260fdd>>. Acesso em: 15 nov 2020.
- SANTOS, Júlio C. **VISEDU:** aplicativo de realidade aumentada usando objetos interativo. 2015a. 67 f. Trabalho de Conclusão de Curso (Ciências da Computação) – Universidade Regional de Blumenau, Blumenau.
- SHAIKH, Raqueeb. **OpenCv Perspective Transformation.** [S.I.], [2020]. Disponível em: <<https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143>>. Acesso em: 29 out. 2020.



TUTORIALSPPOINT, **Perspective Transformation**. [S.1.], [2013]. Disponível em:
<https://www.tutorialspoint.com/dip/perspective_transformation.htm>. Acesso em: 29 out. 2020.

UNITY. **Color**. [S.1.], [2014]. Disponível em: <<https://docs.unity3d.com/ScriptReference/Color.html>>. Acesso em: 27 out. 2020.

