UNIVERSIDADE FEDERAL DE SANTA CATARINA

MAQUIAR: SOLUÇÃO COM REALIDADE AUMENTADA APLICADA NO E-COMMERCE DE MAQUIAGEM

MATEUS NUNES CECHETTO

Florianópolis - SC
2021-2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

MAQUIAR: UMA SOLUÇÃO COM REALIDADE AUMENTADA APLICADA NO E-COMMERCE DE MAQUIAGEM

MATEUS NUNES CECHETTO

Trabalho Conclusão do Curso de Graduação apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Florianópolis - SC
2021-2

Mateus Nunes Cechetto

MaquiAR: Uma solução com Realidade Aumentada aplicada no e-commerce de maquiagem

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação

Banca Examinadora:

_____
Prof. José Eduardo De Lucca
Universidade Federal de Santa Catarina
Orientador

_____
Prof. Dr. Frank Augusto Siqueira
Universidade Federal de Santa Catarina
Avaliador

_____
Prof. Dr. Alexandre Augusto Biz
Universidade Federal de Santa Catarina
Avaliador

RESUMO

Compras online estão cada vez mais populares, e receberam um impulso ainda maior com a pandemia da COVID-19. Assim, foram evidenciados tanto suas vantagens em relação ao comércio em lojas físicas, como a comodidade, quanto suas desvantagens. Uma delas é a impossibilidade de tocar o produto com as mãos. É difícil ter certeza quanto de como o produto é no mundo real já que a maior parte dos produtos são apresentados em fotos 2D e breve descrição. Com isso muitos consumidores acabam desistindo da compra ou se arrependendo dela porque o produto não era o esperado. O objetivo deste trabalho é desenvolver um aplicativo com Realidade Aumentada de maquiagem para dispositivos móveis, com o propósito de levar aos usuários uma experiência diferente na apresentação de maquiagem online e maior segurança para comprá-las. Para sustentar a proposta do aplicativo, foi estudado na literatura os impactos do uso de Realidade Aumentada na intenção de compra dos consumidores e foram analisados aplicativos populares relacionados ao comércio eletrônico de maquiagens. A proposta deste trabalho é um aplicativo Android que permitirá os usuários experimentarem diferentes tipos de maquiagem através da câmera do smartphone, por meio de filtros com Realidade Aumentada e terá acesso a informações das maquiagens que estão experimentando e de como comprá-las. É descrito o processo de desenvolvimento do aplicativo, a arquitetura utilizada e as dificuldades encontradas. Por fim, são apresentados testes realizados com usuários e os seus resultados.

Palavras-chave: realidade aumentada; comércio eletrônico; desenvolvimento para dispositivos móveis; Android; maquiagem.

ABSTRACT

Online shopping is increasingly popular, and has received a boost with the COVID-19 pandemic. Thus, both its advantages in relation to shopping in physical stores, such as convenience, and its disadvantages were evidenced. One of the disadvantages is the impossibility to touch the product with your hands. It is hard to evaluate how the product looks in real world since most of the products are presented in 2D photos and a brief description. As a result, many consumers end up giving up on the purchase or regretting it because the product was not what they expected it to be. The goal of this work is to develop a makeup Augmented Reality application for mobile devices, with the purpose of bringing users a different experience in how makeup is presented online and greater security to buy them. To support this application's proposal, the impacts of the use of Augmented Reality on consumers' purchase intentions were studied in the literature and popular applications related to makeup e-commerce were analyzed. The proposal of this work is an Android application that will allow users to try different types of makeup through the smartphone's camera, through filters with Augmented Reality and the users will have access to information about the makeup they are trying and how to buy them. The application development process, the architecture used and the difficulties encountered are described. Finally, tests performed with users and their results are presented.

Keywords: augmented reality; e-commerce; mobile development; Android; makeup.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
GLSL	*OpenGL Shading Language*
LiDAR	*Light Detection And Ranging*
MVC	*Model-View-Controller*
PDP	Página de detalhes dos produtos
RA	Realidade Aumentada
SaaS	*Software as a Service*
SDK	*Software Development Kit*
SUS	*System Usability Scale*
VDR	*Virtual Dressing Room*
VFR	*Virtual Fitting Room*
VPN	*Virtual Private Network*

SUMÁRIO

1. INTRODUÇÃO

E-commerce, também conhecido como comércio eletrônico, “refere-se a um modelo de negócio que permite empresas e pessoas comprarem e venderem produtos e serviços pela internet”. (BLOOMENTHAL, 2021 tradução nossa). No e-commerce, ao invés do comprador ir fisicamente à loja, olhar o produto, decidir se vai ou não comprá-lo, pagar, caso tenha optado em comprar, ele faz todo esse processo virtualmente. O *e-commerce* tem como uma de suas principais vantagens a comodidade que ele traz ao consumidor, pois funciona ininterruptamente, ao contrário de lojas físicas que possuem horário comercial, apresenta ao consumidor uma maior variedade de produtos e o consumidor não precisa sair de casa para realizar a compra. (BLOOMENTHAL, 2021)

Essa comodidade é valorizada pelo consumidor brasileiro. (E-COMMERCE BRASIL, 2021). Segundo dados da 44ª edição do Webshoppers, relatório sobre *e-commerce* elaborado pela Ebit | Nielsen, as vendas no *e-commerce* no Brasil atingiram R$ 53,4 bilhões no primeiro semestre de 2021, crescimento de 31% em relação ao mesmo período de 2020. O relatório aponta também um crescimento no número de pedidos, atingindo a marca de 100 milhões de pedidos, e do ticket médio, que aumentou 22%. Além disso, foram relatados 6,2 milhões de novos consumidores. Ressalta-se que o primeiro período de 2020 foi um período de crescimento acentuado no *e-commerce*, devido à pandemia da COVID-19, então os dados de 2021 que demonstraram crescimento em relação àqueles, explicitam a forte adesão do consumidor brasileiro ao *e-commerce*. Outro dado relevante extraído do mesmo relatório dá conta que metade das vendas do *e-commerce* ocorreram via dispositivos móveis. (E-COMMERCE BRASIL, 2021).

O *e-commerce*, porém, também possui desvantagens. A segurança é uma das principais delas. (SAMPAIO, 2019). Tanto o consumidor necessita da certeza de que o *site* no qual está colocando seus dados é legítimo e seguro, quanto a empresa precisa da certeza de que o usuário é uma pessoa real, sem intenções de fraudes. Essa certeza é necessária pois há inúmeras atividades fraudulentas na internet. Mais uma desvantagem é o tempo de entrega, (SAMPAIO, 2019) já que o comprador não sai com o produto em mãos como no comércio tradicional. Além disso, há

clientes que preferem “sentir” o produto com as próprias mãos; como por exemplo verificar o caimento ou a combinação de roupas, calçados e acessórios. Este é um problema que este trabalho irá tentar abordar, ou seja, a impossibilidade de visualizar um produto aplicado numa situação real.

Em relação ao último ponto citado, a maioria dos produtos comercializados online são apresentados ao consumidor por uma breve descrição, fotos e avaliações de outros compradores. No escopo deste trabalho, essa forma de apresentação do produto será denominada de “forma tradicional”. Muitas vezes, porém, essas informações não são suficientes para que a decisão do consumidor em comprar ou não o produto seja a ideal. Com isso, um dos seguintes cenários ocorre: o cliente desiste da compra, o cliente busca em uma loja física o produto, o cliente compra o produto e acaba insatisfeito, necessitando devolução ou troca, ou o cliente compra o produto e fica satisfeito. Apenas o último cenário é ideal, tanto para o consumidor quanto para o vendedor.

O primeiro cenário é um cenário muito comum, segundo o Zero Friction Future de 2019, estudo desenvolvido pelo Facebook, 46% dos brasileiros desistiram de compras online em 6 meses, e o estudo aponta a insuficiência de informações do produto como um dos principais pontos de atrito durante a fase de descoberta. (ZERO FRICTION FUTURE, 2019).

O segundo cenário também é prejudicial, tanto para o consumidor, pois retira a comodidade do *e-commerce*, quanto para o vendedor, pois o cliente, ao estar numa loja física, pode optar por comprar o produto nela, com pronta entrega.

E o terceiro cenário é um pesadelo, para ambos. O consumidor, além de insatisfeito por não ter o produto desejado, ainda precisa entrar em contato com o suporte, reembalar o produto, ir até o operador de logística designado pela loja (que pode ser uma agência dos Correios ou escritório de transportadora), e ainda esperar ou a devolução do dinheiro ou o novo produto. O vendedor, além de ter que lidar com o processo de troca ou devolução, pode ter seu relacionamento com o cliente abalado. E este cenário é muito mais comum do que o desejado, de acordo com uma pesquisa realizada pela Invesp, 30% de todos os produtos comprados online são devolvidos, um número muito maior que os 8,89% das lojas físicas, sendo o

principal motivo para a diferença a incapacidade de sentir o produto com as mãos. (SALEH, 2022)

Este trabalho apresenta uma proposta para abordar esse problema, através do uso de realidade aumentada (RA) na apresentação do produto ao consumidor, no caso deste trabalho, os produtos são maquiagens. Resumidamente, RA é a sobreposição de objetos virtuais ao mundo real do usuário. Ela se difere da realidade virtual, que tem como definição a criação de um mundo virtual no qual o usuário é inserido em tempo real, por meio de um avatar. Desta forma, a RA é mais benéfica do que a realidade virtual no contexto de *e-commerce* porque ela melhora a compreensão dos consumidores sobre os produtos, e proporciona o prazer de se verem usando o item sem precisarem ir à loja física. (YIM; CHU; SAUER, 2017). Assim, o uso de RA no *e-commerce* promete melhorar a forma como um produto é apresentado ao consumidor e por conseguinte: melhorar a percepção e experiência do usuário em relação ao produto, aumentar a intenção de compra e diminuir as devoluções.

1.1 Objetivo Geral

Este trabalho tem como objetivo geral desenvolver um aplicativo móvel com realidade aumentada de maquiagem. Neste aplicativo, as maquiagens serão apresentadas ao usuário de uma forma diferente da tradicional, proporcionando-lhe uma experiência diferente e maior segurança para adquiri-las.

1.2 Objetivos Específicos

- Analisar as vantagens da utilização de realidade aumentada para apresentar um produto ao usuário
- Examinar a viabilidade, tanto tecnológica quanto comercial, da utilização de realidade aumentada em diferentes tipos de produto.
- Conhecer o estado da arte em realidade aumentada e suas aplicações no *e-commerce*.
- Aprofundar o conhecimento no desenvolvimento de aplicativos móveis.

2. ESTUDOS REALIZADOS

Este capítulo apresenta uma revisão teórica dos conceitos fundamentais para a compreensão deste trabalho, bem como uma análise da literatura a respeito da realidade aumentada e sua aplicação no comércio eletrônico. Apresenta também o estado da arte no desenvolvimento de realidade aumentada para dispositivos móveis.

2.1 Realidade Aumentada (RA)

De acordo com Azuma (1997 apud SILVA; ABREU; TEIXEIRA, 2021), realidade aumentada pode ser definida como uma variação da Realidade Virtual. Eles afirmam que a realidade virtual imerge o usuário completamente em um ambiente virtual, e o usuário, enquanto imerso, não consegue ver o mundo real. Em contrapartida, a RA permite ver o mundo real, com objetos virtuais sobrepostos e coexistindo com o mundo real. O autor completa dizendo que a RA complementa o mundo real, ao invés de substituí-lo, como faz a Realidade Virtual.

Lu e Smith (2007) definem realidade aumentada como uma tecnologia que consegue misturar ou sobrepor objetos virtuais ao mundo real. Ao contrário da Realidade Virtual que substitui o mundo físico por um virtual, a RA aprimora a realidade ao integrar objetos virtuais ao mundo físico. Esses objetos virtuais se tornam, de certa forma, uma parte igual do ambiente real. (LU; SMITH, 2007).

Segundo Carmigniani et al. (2011 apud SILVA; ABREU; TEIXEIRA, 2021), a RA pode ser vista como uma tecnologia interativa, sendo uma visão direta ou indireta de um ambiente físico que foi complementado pela adição de informações virtuais.

O objetivo da RA pode ser descrito como permitir ao seu utilizador uma interação com o ambiente real e com os elementos virtuais sobrepostos no mesmo, de uma forma natural e intuitiva, sem necessidade de uma grande adaptação por parte deste. (TORI; HOUNSELL, 2018 apud SILVA; ABREU; TEIXEIRA, 2021).

2.2 *Virtual try-on* e *Virtual Fitting Room*

Virtual try-on, ou apenas *try-on*, é uma tecnologia que pode entregar ao

comprador informações de um produto semelhantes às informações obtidas pela experimentação direta do produto. *Try-on* permite aos consumidores ver os produtos por vários ângulos, dar zoom, rotacionar o produto e vê-lo em diferentes cores, em um modelo virtual que imita suas aparências. (KIM; FORSYTHE, 2008). Segundo Kim e Forsythe (2008) a interatividade e o envolvimento do consumidor criados pelo uso da tecnologia podem melhorar a experiência da compra online.

De acordo com Wagner (2007 apud KIM; FORSYTHE, 2008), *virtual try-on* está se tornando mais usado no comércio eletrônico de varejo pois diminui a diferença da experiência entre compras online e offline. O mesmo autor afirma ainda que adicionar tecnologias interativas como *virtual try-on* ao comércio eletrônico aumenta significantemente as taxas de conversão de um website.

Embora o conceito de *try-on* tenha sido criado antes de suas aplicações com realidade aumentada, é evidente que o uso da RA potencializa o *try-on* virtual de produtos, pois permite que, ao invés de utilizar modelos, o consumidor se veja utilizando o produto, melhorando ainda mais sua experiência.

Virtual Fitting Room (VFR), também conhecida como *Virtual Dressing Room* (VDR), é uma tipo de tecnologia de realidade aumentada que sobrepõe imagens virtuais de roupas e acessórios sobre uma imagem em tempo-real do consumidor com o objetivo de criar o efeito de um espelho, assim como os de cabines de provadores de lojas físicas. A tecnologia serve não apenas para o consumidor ter maior precisão quanto ao tamanho certo da roupa como também os permite se ver utilizando as roupas, potencialmente aumentando sua confiança em realizar a compra online. VFR ajuda a criar uma experiência mais realista do produto e gera maior imersão e engajamento na experiência da compra online. (YAOYUNEYONG et al., 2018). Podemos afirmar, então, que VFR é um exemplo de *try-on*.

2.3 Trabalhos Relacionados

Esta seção visa apresentar alguns trabalhos relacionados ao uso de realidade aumentada na apresentação de produtos do comércio eletrônico e seu impacto no processo de decisão de compra do consumidor. Na fase de pesquisas, foi utilizado o *Google Scholar*, com os seguintes termos e suas traduções para o

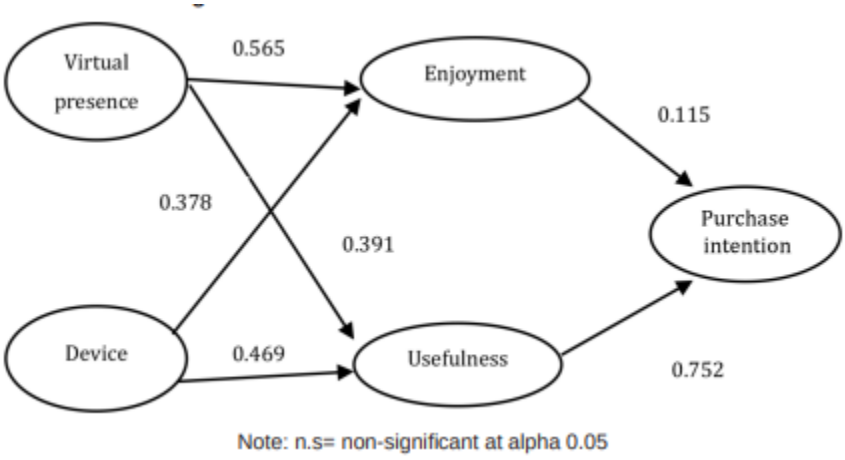
inglês: “realidade aumentada comércio eletrônico", “realidade aumentada varejo”, “*try-on* virtual comércio eletrônico”, “*try-on* virtual de maquiagem”, “*try-on* virtual varejo” e “impacto da realidade aumentada no comércio eletrônico”. Foi dada a preferência para artigos recentes, de 2018 para cá. Chegou-se, assim, a 9 artigos considerados relevantes. Após a leitura, realizou-se uma avaliação inicial dos artigos e 3 foram considerados mais apropriados para este estudo e foram selecionados, analisados e comparados entre si. Para cada artigo, será apresentado um pequeno resumo, listando os objetivos gerais, o desenvolvimento e os resultados e conclusões.

2.3.1 The Effect of Augmented Reality Shopping Applications on Purchase Intention

Este artigo (LEONNARD; PARAMITA; MAULIDIANI, 2019) apresenta um estudo realizado para encontrar o impacto do uso de realidade aumentada no comércio eletrônico na intenção de compra. Para isso, são utilizados os termos: presença virtual, que é um indicador para avaliar o nível de realidade dos objetos virtuais oferecidos pela RA, dispositivo, que é o meio pelo qual a pessoa vai ter a experiência da RA, utilidade, que é um indicador para avaliar o quão efetiva e eficiente a tecnologia é para ajudar os consumidores a encontrar as informações que precisam para avaliar os produtos, e prazer, que mede o nível de satisfação e prazer que o consumidor tem ao utilizar a tecnologia. Os dois primeiros medem a qualidade da realidade aumentada, enquanto os dois últimos medem a experiência do consumidor.

O método utilizado foi uma pesquisa com 89 consumidores de comércio eletrônico, com idades entre 15 e 35 anos de Jacarta do Sul, Indonésia. A pesquisa constituía em 19 afirmações e foi utilizada a escala Likert de 5 pontos. Foram utilizados métodos estatísticos para encontrar os efeitos de e para cada indicador. O resultado é representado pela figura abaixo:

Figura 1 - Resultado do estudo de Leonnard, Paramita e Maulidiani



Fonte: LEONNARD; PARAMITA; MAULIDIANI, 2019

Os resultados mostram que a presença virtual e o dispositivo diretamente afetam significantemente a utilidade e o prazer e indiretamente afetam a intenção de compra. A utilidade afeta fortemente a intenção de compra, enquanto o prazer não afeta de forma significativa a intenção de compra. É importante ressaltar que o estudo foi realizado majoritariamente com estudantes de ensino médio e superior, que são mais receptivos a novas tecnologias.

2.3.2 I virtually try it … I want it ! Virtual Fitting Room: A tool to increase on-line and off-line exploratory behavior, patronage and purchase intentions

Este artigo (BECK; CRIÉ, 2018) tem como objetivo encontrar os impactos que o uso de *Virtual Fitting Room* pode trazer para uma loja, tanto em relação às compras na loja virtual quanto física. Para isso, são utilizados como indicadores: curiosidade, patronagem e intenção de compra. O artigo é dividido em duas partes principais, o *background* teórico e as pesquisas, resultados e conclusões.

No *background* teórico são explicados os conceitos e são criadas as hipóteses, que serão investigadas na etapa da pesquisa. Foram analisados: a curiosidade do consumidor como uma variável motivacional e sua ligação com VFR, os efeitos de VFR nas intenções de patronagem do consumidor e os efeitos de VFR nas intenções de compra. Assim, foram criadas 7 hipóteses:

- H1: A presença de VFR em um website aumenta a curiosidade específica perceptiva sobre o produto

- H2: A presença de VFR em um website aumenta a intenção de patronagem de um vendedor online
- H3: A presença de VFR em um website aumenta a intenção de patronagem de uma loja física
- H4: A presença de VFR em um website aumenta a intenção de compra online
- H5: A presença de VFR em um website aumenta a intenção de compra offline
- H6: A curiosidade específica perceptiva sobre o produto e a intenção de patronagem do website medeiam o efeito da presença de VFR na intenção de compra online
- H7: A curiosidade específica perceptiva sobre o produto e a intenção de patronagem da loja física medeiam o efeito da presença de VFR na intenção de compra offline

Foram realizados 2 experimentos, um a fim de examinar os efeitos de VFR nas compras online (hipóteses H1, H2, H4 e H6) e outro para as offline (hipóteses H1, H3, H5 e H7). No primeiro, 228 estudantes de universidades europeias foram separados aleatoriamente em dois grupos. Um dos grupos recebeu um website com um catálogo e o outro com VFR, ambos com os mesmos modelos de roupas. Depois de explorar o website, os participantes responderam a um questionário. Foram utilizadas escalas conhecidas para mensurar a curiosidade específica perceptiva, a intenção de patronagem e a intenção de compra, todas com escala Likert de 7 pontos. O segundo estudo foi composto por 241 consumidores que também foram separados aleatoriamente em dois grupos. Um grupo recebeu um website com um catálogo e o outro com VFR, ambos com os mesmos modelos de óculos. Depois de explorar o website, os participantes responderam a um questionário. Foram utilizadas as mesmas escalas do primeiro estudo.

O experimento 1 suportou as hipóteses H1, H2 e H4, mostrando que a presença de VFR em um website aumenta significativamente a curiosidade sobre o produto, a intenção de patronagem e a intenção de compra, quando comparada a um catálogo online. H6 também é validada nesse experimento, mostrando que ao

aumentar a curiosidade sobre o produto, VFR diretamente aumenta as intenções de patronagem e compra.

O experimento 2 mostra que a curiosidade, a patronagem da loja física e a intenção de compra offline são positivamente afetadas pela presença de VFR no website, indo de acordo com as hipóteses H1, H3 e H5. A H7 foi parcialmente validada pelo experimento 2.

Em suma, o estudo mostra que o uso de VFR em um website aumenta a curiosidade do consumidor, as chances de patronagem e de compra do produto, tanto para lojas virtuais quanto físicas. Nota-se que foram utilizados dois contextos diferentes (lojas virtuais e físicas), duas categorias de produtos diferentes (roupas e óculos) e duas amostras diferentes (estudantes e consumidores), o que aumenta a confiabilidade dos estudos deste artigo.

2.3.3 Is Augmented Reality Technology an Effective Tool for E-commerce? An Interactivity and Vividness Perspective

O artigo (YIM; CHU; SAUER, 2017) avalia a eficácia do uso de realidade aumentada como uma ferramenta de e-commerce. Para isso, foram realizados dois estudos: Estudo 1, que compara a avaliação dos consumidores da apresentação de produtos usando realidade aumentada e a avaliação dos consumidores da apresentação de produtos usando tecnologias web tradicionais; e Estudo 2, que compara os caminhos pelos quais os consumidores avaliam os produtos com um foco em interatividade e nitidez, também comparando AR e web.

O artigo relata que cada vez mais as empresas estão adotando diferentes formas de realidade aumentada para fornecer aos seus consumidores uma experiência mais realista de seus produtos. Porém, nenhuma evidência prévia a este artigo tinha confirmado que RA é uma ferramenta de persuasão melhor do que os meios existentes de apresentar os produtos online.

Ambos os estudos foram feitos usando tecnologias de AR e web tradicionais existentes no mercado, os produtos avaliados foram óculos de sol e relógios e os participantes foram estudantes de ensino superior dos Estados Unidos. O Estudo 1 revelou que produtos apresentados com RA são geralmente superiores aos

apresentados de maneira tradicional na web em *media novelty*, imersão, *media enjoyment*, utilidade, *attitude toward medium* e intenção de compra. O Estudo 2 confirma que interatividade e nitidez geram avaliações positivas pelo consumidor através do aumento da interatividade. O Estudo 2 também mostra que avaliações positivas de RA estão diretamente relacionadas a efeitos de novidade, e que existe uma correlação negativa entre experiência prévia no uso da tecnologia e sua avaliação nas avaliações de produtos apresentados com RA, o que é consistente com a teoria da habituação-tédio (habituation–tedium theory), o que não aconteceu nas avaliações dos produtos apresentados pelas formas web tradicionais.

O Estudo 2 também possuía uma questão aberta, opcional, na qual os participantes deixavam sua opinião sobre a tecnologia RA. Foram utilizados classificadores e analisadores de texto. As palavras mais frequentes foram "*technology*", "*cool*", "*fun*" e "*new*". Palavras relacionadas a compras, como "*buy*", "*try*" e "*store*" também foram destacadas. O artigo afirma que a percepção dos consumidores é geralmente positiva quanto ao uso da realidade aumentada na apresentação dos produtos mas que ela também possui algumas críticas, como falta de desempenho, usuários afirmaram que tiveram dificuldade de posicionar os óculos em seus rostos, e falta de qualidade nos gráficos, usuários afirmaram que não pareciam muito realistas.

2.4 Desenvolvimento de Realidade Aumentada

Nesta seção será apresentado o estado da arte do desenvolvimento de realidade aumentada para dispositivos móveis.

2.4.1 Kits de desenvolvimento de software de Realidade Aumentada

Os *software development kits* (SDKs) de realidade aumentada são conjuntos de ferramentas que fornecem aos desenvolvedores um ambiente de desenvolvimento onde eles são capazes de criar e implementar todas as funcionalidades que irão compor o *core* das aplicações com capacidade de AR. Dentre as funcionalidades providas pelos SDKs, podemos citar:

- Reconhecimento do ambiente através da câmera e de sensores

- *Tracking*: rastreamento de objetos do ambiente e seus movimentos
- Renderização dos objetos virtuais no mundo real em tempo real

2.4.1.1 ARKit

ARKit é o SDK da Apple e foi lançado em junho de 2017. É um kit voltado a iOS e permite a criação de aplicativos com realidade aumentada para iPhones e iPads com iOS 11 ou superior. O ARKit é gratuito, mas para ter acesso a ele é necessário uma conta de desenvolvedor Apple. O ARKit está em sua 5ª versão, lançada em junho. Dentre as funcionalidades do ARKit 5, podemos destacar:

- Rastreamento de múltiplos rostos ao mesmo tempo (introduzido no ARKit 3)
- *People occlusion* - que faz com que objetos sejam renderizados tanto a frente quanto atrás das pessoas (também introduzido no ARKit 3)
- Escanear objetos reais e criar modelos 3D que podem ser usados nas aplicações de RA.

Sobre essa última funcionalidade, ela foi adicionada no ARKit 5 e é possível devido aos *scanner Light Detection And Ranging* (LiDAR) e às câmeras nos dispositivos iOS 15. Com eles, é possível escanear um objeto e transformá-lo em um arquivo USDZ, que pode ser importado e incorporado aos projetos dos aplicativos como um modelo 3D.

As informações citadas sobre o ARKit foram adquiridas no site oficial da Apple. (APPLE, 2022).

2.4.1.2 ARCore

ARCore é o SDK da Google voltado ao desenvolvimento de RA e foi lançado em março de 2018 como o sucessor do Projeto Tango, que tivera seu início em 2014. O kit possui suporte tanto para dispositivos Android quanto para dispositivos iOS. O ARCore é totalmente gratuito e de código aberto. Segundo o site oficial (GOOGLE, 2022a) os conceitos fundamentais do ARCore são:

- Rastreamento de movimento: entende onde o telefone está em relação ao seu redor.
- Compreensão do ambiente: detecta pontos e planos no mundo real e disponibiliza essas informações para o aplicativo, que pode usar essas informações para colocar objetos virtuais em superfícies planas.
- Compreensão de profundidade: cria mapas de profundidade. As informações desse mapa podem ser usadas para criar experiências mais imersivas e realistas, como fazer objetos virtuais colidirem com superfícies reais ou fazê-los aparecer na frente ou atrás de objetos do mundo real.
- Estimativa de luz: detecta informações sobre a iluminação do ambiente e fornece correção de cor, dessa forma é possível iluminar os objetos virtuais nas mesmas condições do ambiente ao seu redor, aumentando a sensação de realismo.
- Pontos orientados: permite colocar objetos virtuais em superfícies angulares.
- Âncoras e rastreáveis: permite que os objetos se mantenham na posição mesmo movendo a câmera ao longo do tempo. Ao colocar um objeto virtual, é necessário definir uma âncora para garantir que o objeto seja rastreado. Planos e pontos são um tipo especial de objeto chamado rastreável. É possível ancorar objetos virtuais a rastreáveis específicos para garantir que a relação entre o objeto virtual e o rastreável permaneça estável mesmo enquanto o dispositivo se move.
- Imagens aumentadas: permite criar aplicativos RA que podem responder a imagens 2D específicas, como embalagens de produtos que quando apontados pela câmera mostram detalhes do produto ou pôsteres de filmes que quando apontados pela câmera fazem um personagem do filme aparecer e encenar uma cena.
- Compartilhamento: a partir da API ARCore Cloud Anchor, permite criar aplicativos colaborativos ou multijogador. Um dispositivo envia uma âncora e *feature points* próximos à nuvem para hospedagem. Essas âncoras podem ser compartilhadas com outros usuários no mesmo ambiente, assim os aplicativos podem renderizar os mesmos objetos 3D anexados a essas

âncoras, permitindo que os usuários tenham a mesma experiência de AR simultaneamente.

2.4.1.3 Vuforia Engine

Vuforia Engine é um dos SDK para aplicações móveis de realidade aumentada mais usadas no mundo, foi lançado em 2011 e em 2015 foi adquirido pela PTC. (PTC, 2017). As principais funcionalidades do Vuforia Engine podem ser divididas em 3 categorias: *Tracking Images, Tracking Objects* e *Tracking Environments.* Segue as funcionalidades de cada categoria:

Tracking Images:

- *Image target:* permite anexar conteúdos a imagens planas, como mídia impressa e embalagem de produtos
- *Multi-targets:* permite usar mais de um *Image target* e organizá-los em formas geométricas regulares ou em qualquer disposição arbitrária de superfícies planas com vários alvos.
- *Cylinder targets:* permite reconhecer imagens envolvidas em objetos cilíndricos ou de forma similar (como garrafas de bebida, xícaras de café ou latas de refrigerante).
- VuMarks: são marcadores personalizados que podem codificar uma variedade de formatos de dados. Eles suportam identificação e rastreamento exclusivos para aplicativos de RA.

Tracking Objects:

- *Model targets:* permite reconhecer objetos pelo formato usando modelos 3D pré-existentes.

Tracking environments:

- *Area targets:* permite criar experiências RA nos ambientes através do *Vuforia Area Target Creator App* ou usando um *scanner* 3D do mercado.
- *Ground plane:* permite posicionar conteúdo em superfícies horizontais.

O Vuforia Engine possui um plano básico gratuito e um plano *premium*. O plano básico possui *Image targets, VuMarks, Multi-targets, Cylinder targets* e *Ground plane*, além de 20 *Model targets* e 20 *Area targets* para uso de prototipagem (uso

não comercial). O plano *premium* além de possuir as funcionalidades *Model targets* e *Area targets* por completo, possui suporte à produção. O Vuforia Engine não possui funcionalidades para trabalhar com rostos, portanto não se encaixa no contexto de maquiagens deste trabalho.

As informações citadas foram obtidas no site oficial do Vuforia (VUFORIA, 2022a, 2022b).

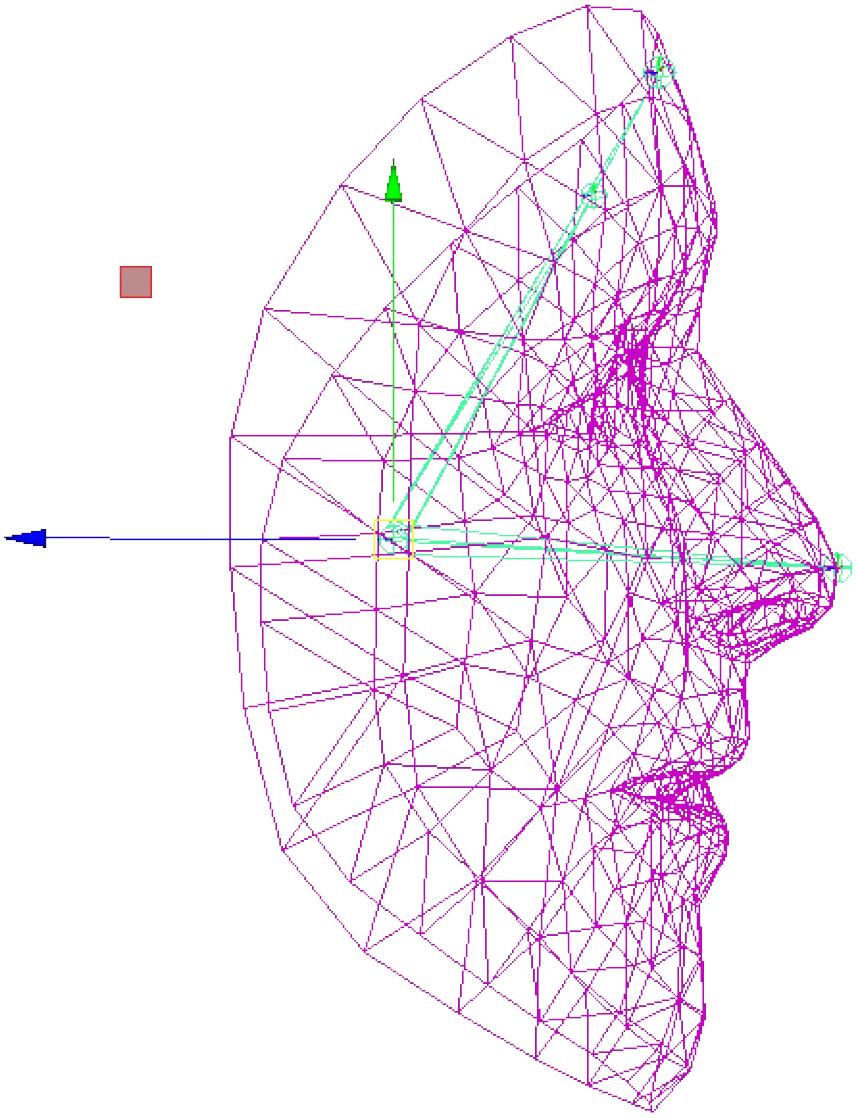
2.4.2 Reconhecimento de rostos

Como o objetivo deste trabalho é o desenvolvimento de um aplicativo de maquiagem, é preciso entender como as tecnologias de realidade aumentada funcionam para reconhecer e rastrear rostos. Assim, as maquiagens que serão desenvolvidas aparecerão na posição correta do rosto do usuário e acompanharão seus movimentos. Esta seção tem como foco as características do desenvolvimento com ARCore, tecnologia escolhida para o desenvolvimento do aplicativo.

O ARCore possui uma *Application Programming Interface* (API) específica para o desenvolvimento de aplicativos de Realidade Aumentada que desejam renderizar objetos virtuais, no caso deste trabalho os filtros de maquiagem, sobre rostos humanos. Esta API é chamada de *Augmented Faces* e fornece pontos de recurso que permitem que o aplicativo identifique automaticamente diferentes regiões de um rosto. (GOOGLE, 2022d).

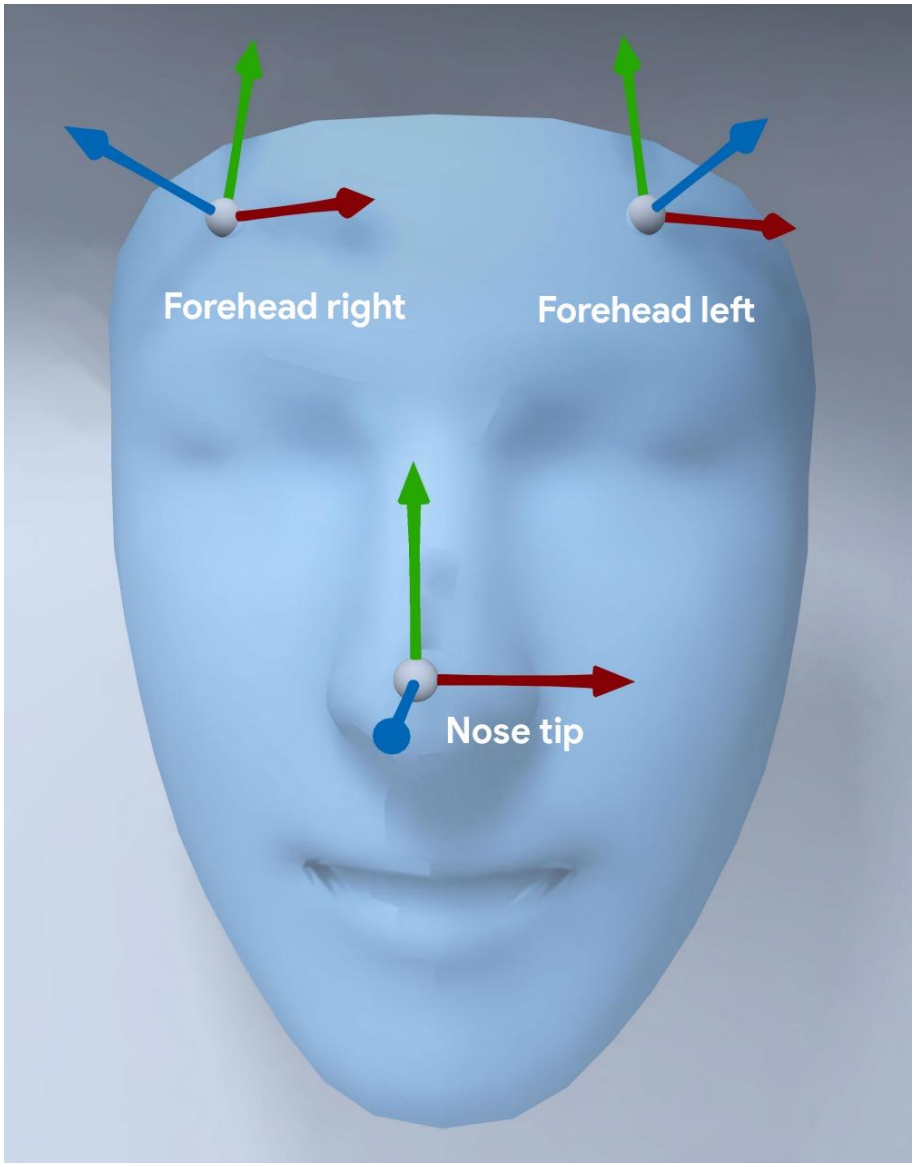
Um “rosto aumentado” é composto por três partes: um ponto central, três pontos de região e uma malha facial 3D. O ponto central localiza-se atrás do nariz e marca o meio da cabeça do usuário. É utilizado para para renderizar objetos como chapéus e bonés, portanto não muito relevante no contexto deste trabalho. Os pontos de região ficam localizados um na parte da esquerda da testa, um na direita, e outro na ponta do nariz e são utilizados para renderizar objetos no nariz e ao redor das orelhas. A malha facial 3D possui 468 pontos que permitem criar texturas adaptáveis e detalhadas que seguem com precisão um rosto. É através dela que serão criados os filtros de maquiagens deste trabalho. (GOOGLE, 2022d).

Figura 2 - Ponto central ARCore *Augmented Faces* API



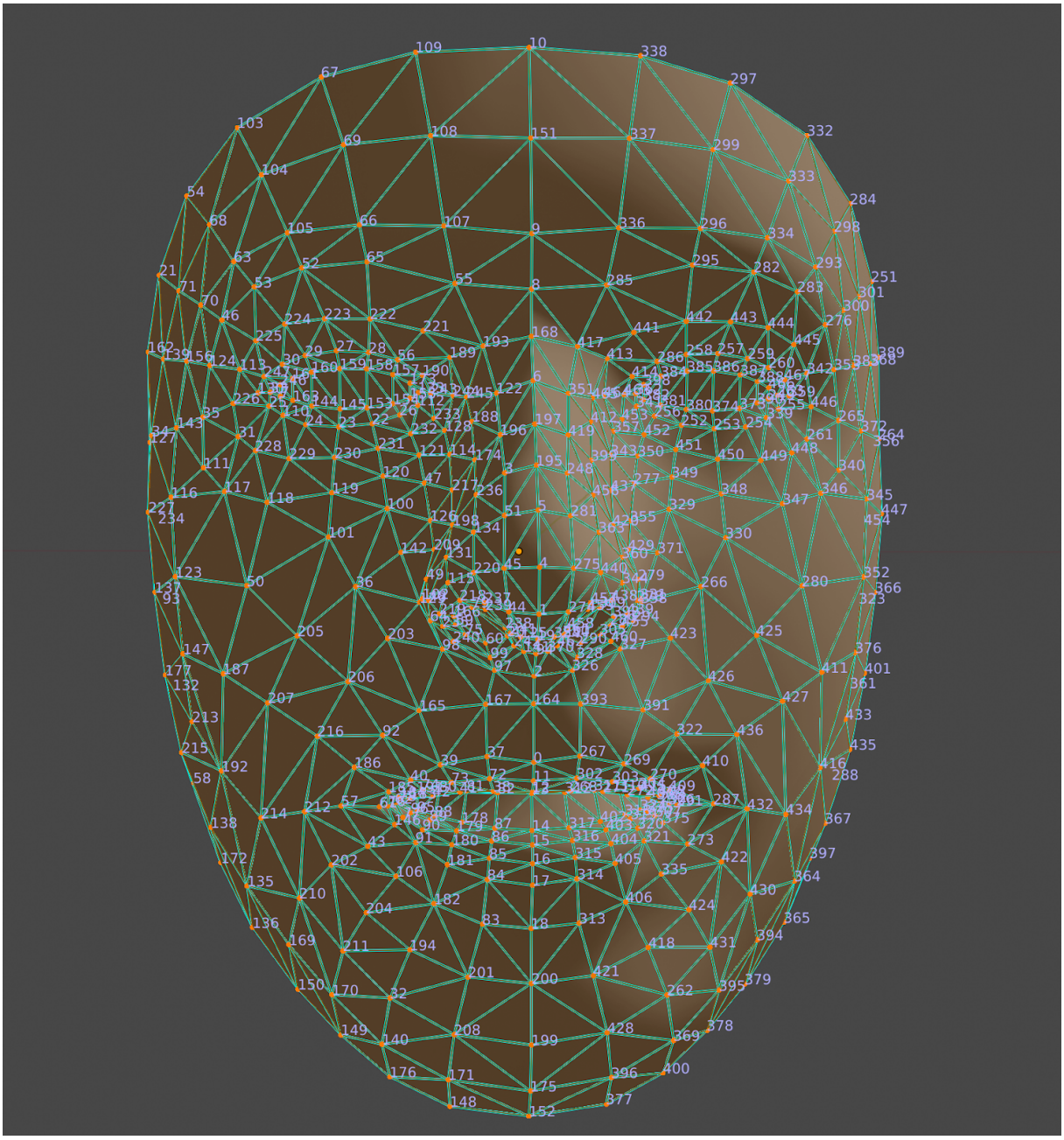
Fonte: Google, 2022

Figura 3 - Pontos de região ARCore *Augmented Faces* API



Fonte: Google, 2022

Figura 4 - Malha facial 3D ARCore *Augmented Faces* API



Fonte: Google, 2022

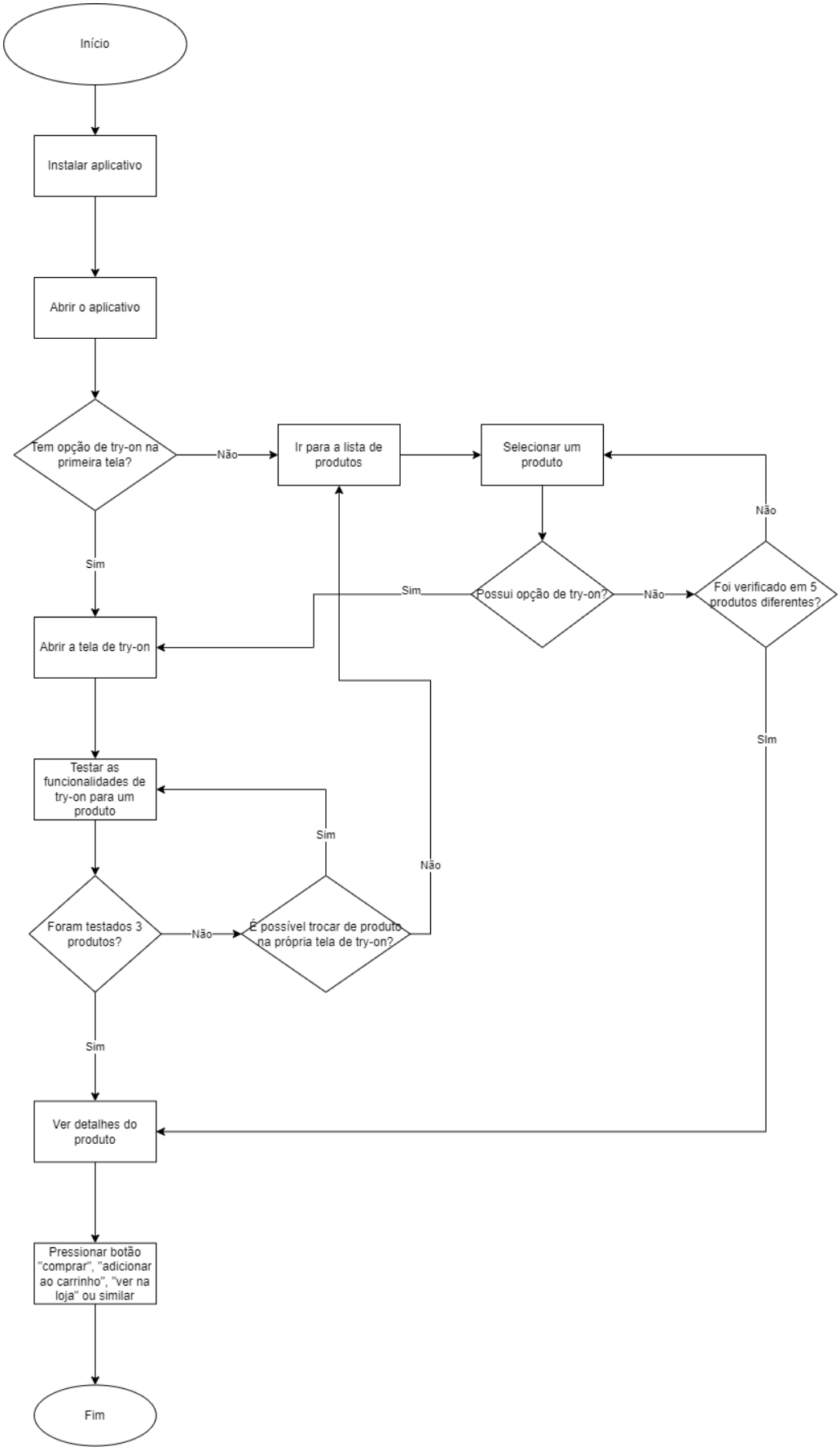
3. APLICATIVOS SEMELHANTES

Com o objetivo de analisar o estado atual do mercado, foi realizada uma pesquisa de aplicativos relacionados a *try-on* e e-commerce de maquiagem. Foram utilizados para a pesquisa os termos-chave: "*try-on* de maquiagem", "experimentar maquiagem", "maquiagem realidade aumentada", "comprar maquiagem", "e-commerce maquiagem" e seus sinônimos em inglês. A pesquisa foi feita na Google Play Store e foram selecionados os aplicativos que possuíam mais downloads. Foram selecionados 8 aplicativos que foram instalados para uma verificação se os mesmos de fato faziam parte do escopo do trabalho. Após a verificação, 3 destes aplicativos foram desconsiderados por serem jogos de maquiagem. Posteriormente, foram adicionados à lista de aplicativos a serem analisados aplicativos de marcas de cosméticos e beleza conhecidos, como o da "Sephora", assim como os aplicativos "Amazon" e "Instagram", que, apesar de não terem aparecido na pesquisa com termos-chave, são *marketplaces* de amplo uso no Brasil e que contém seções de maquiagem.

Com o intuito de avaliar suas características, funcionalidades, vantagens e desvantagens de cada aplicativo, foi traçada a estratégia descrita pelo fluxograma da figura 5. Primeiramente, o aplicativo é instalado. Depois, abre-se o aplicativo e verifica se, logo na tela inicial, possui alguma opção para *try-on*. Caso haja, abre-se a tela de *try-on.* Então, são analisadas as funcionalidades que estão presentes na tela de *try-on* do aplicativo, como por exemplo: mudar a tonalidade de uma maquiagem, aplicar múltiplas maquiagens ao mesmo tempo, tirar foto, etc. A análise dessa tela é feita 3 vezes, com produtos diferentes, para verificar se o aplicativo trata diferentemente as especificidades de diferentes tipos de maquiagem (por exemplo batom e base). Após isso, abre-se a tela de descrição do produto, onde ele é apresentado da maneira tradicional. Então, é pressionado o botão relacionado ao *e-commerce*. Esse o rótulo desse botão provavelmente será "comprar", "adicionar ao carrinho", "ver na loja" ou similar. Assim, analisa-se como o aplicativo lida com a intenção de compra do usuário, se ele possui um e-commerce integrado, se ele direciona para o website da marca do produto ou de um vendedor que possui esse produto, ou se o aplicativo não lida com a intenção de compra - nesse caso o nem

deve existir o botão de comprar. Encerra o fluxo. Caso o aplicativo não possua opção de *try-on* na tela inicial, entra-se na tela de listagem de produtos - seja de forma direta ou através de busca. Se na lista de produtos os produtos que possuírem *try-on* estiverem destacados, um destes produtos destacados será selecionado, caso contrário é selecionado um produto qualquer. É verificado, então, se o produto possui *try-on*. Caso possua, o segue o caminho principal do fluxo a partir da abertura da tela de *try-on.* Caso contrário, são selecionados outros quatro produtos. Se nenhum deles possuir *try-on*, é pulada essa parte do caminho principal, indo direto para a parte de ver detalhes do produto.

Figura 5 - Fluxograma do processo de análise dos aplicativos relacionados


Início
Instalar aplicativo
Abrir o aplicativo
Tem opção de try-on na primeira tela?
Não
Ir para a lista de produtos
Selecionar um produto
Não
Sim
Possui opção de try-on?
Sim
Não
Foi verificado em 5 produtos diferentes?
Abrir a tela de try-on
Sim
Testar as funcionalidades de try-on para um produto
Sim
Não
Foram testados 3 produtos?
Não
É possível trocar de produto na própria tela de try-on?
Sim
Ver detalhes do produto
Pressionar botão "comprar", "adicionar ao carrinho", "ver na loja" ou similar
Fim

Fonte: Autor

Os aplicativos selecionados foram instalados no dia 4 de fevereiro de 2022, em suas versões *free* mais recentes, através da Google Play Store do Brasil, em um Motorola One Hyper com Android versão 11. Após pesquisas na internet, notou-se a possibilidade de analisar aplicativos que não estão disponíveis no Brasil mas que possuem relevância no escopo deste trabalho, com o auxílio de uma *Virtual Private Network* (VPN). Assim, foram instalados os aplicativos “Sephora - Beauty Shopping”, no dia 19 de fevereiro de 2022 através da Google Play Store de Singapura e, no dia 20 de fevereiro de 2022, a versão estadunidense do aplicativo “Amazon” através da Google Play Store dos Estados Unidos.

Os aplicativos foram separados em 4 categorias: aplicativos de e-commerce, aplicativos de *try-on* de maquiagem, aplicativos de e-commerce com *try-on* e aplicativos de *try-on* voltados ao e-commerce. As categorias serão detalhadas em suas respectivas seções.

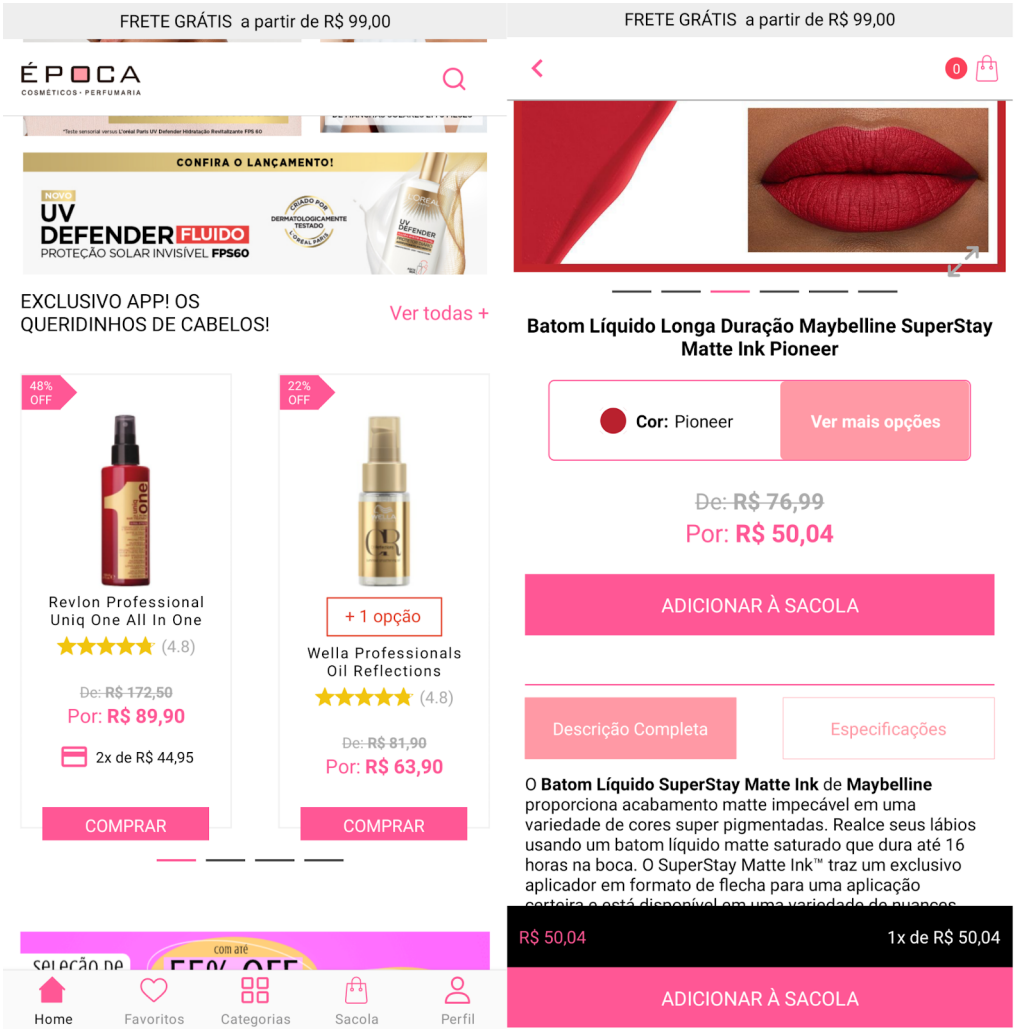
3.1 Aplicativos de e-commerce de maquiagem

Os aplicativos desta categoria realizam o comércio eletrônico de produtos de maquiagem. Neles, os produtos são apresentados ao usuário na forma tradicional: foto do produto, foto da sua cor, fotos de pessoas utilizando o produto, descrição escrita do produto e avaliações de compradores. Em todos os aplicativos analisados o processo de compra é realizado no próprio aplicativo, desde cadastro do usuário até métodos de pagamento e detalhes da entrega. Além dos produtos ofertados e seus preços, não foi encontrada grande diferença entre os aplicativos da categoria.

Os aplicativos analisados que pertencem a esta categoria foram:

- Época Cosméticos e Maquiagens: aplicativo de uma das maiores lojas de e-commerce de perfumes, cosméticos e maquiagens do Brasil, que começou sua atuação no *e-commerce* em 2006 e que desde 2013 faz parte do Grupo Magazine Luiza. (ZUINI, 2013). Possui mais de 1 milhão de downloads na Google Play Store. Link do aplicativo na Google Play Store: https://play.google.com/store/apps/details?id=com.mobfiq.epocacosmeticos&hl=pt_BR&gl=US.

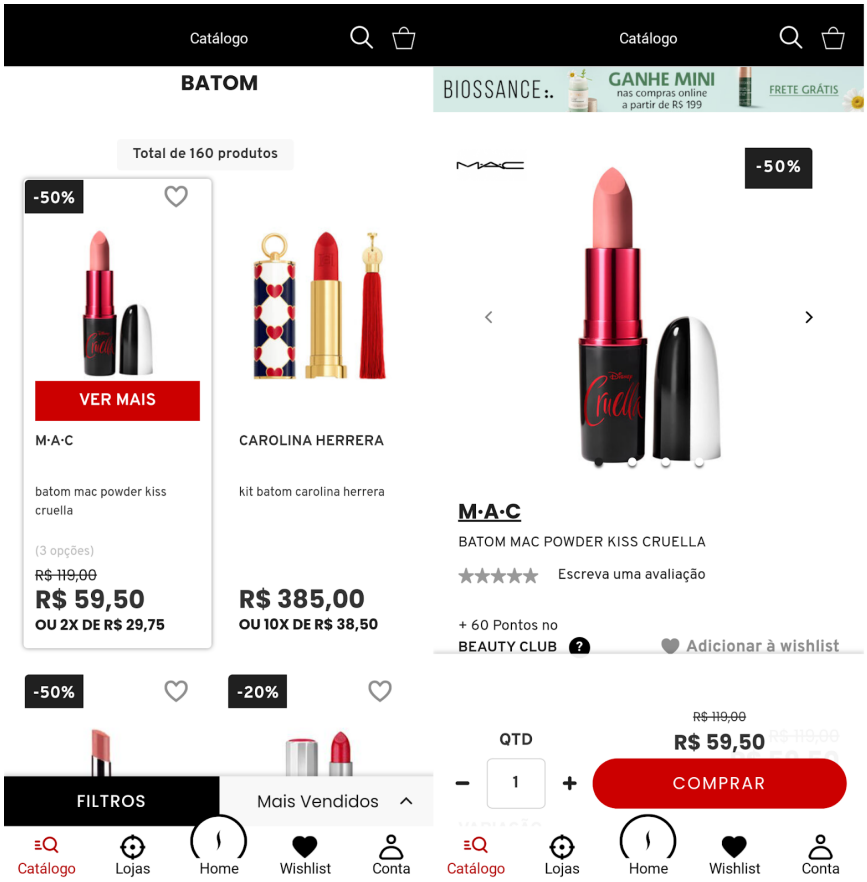
Figura 6 - Telas do Época Cosméticos e Maquiagens



Fonte: Capturas de tela do aplicativo Época Cosméticos e Maquiagens

- Sephora: aplicativo nacional da multinacional francesa que possui mais de 2700 lojas em 35 países e que para cada país ou região possui seu próprio aplicativo, o aplicativo do Brasil possui mais de 50 mil downloads na Google Play Store. (SEPHORA, 2022). Link do aplicativo na Google Play Store: https://play.google.com/store/apps/details?id=latam.sephora.sephorabr.

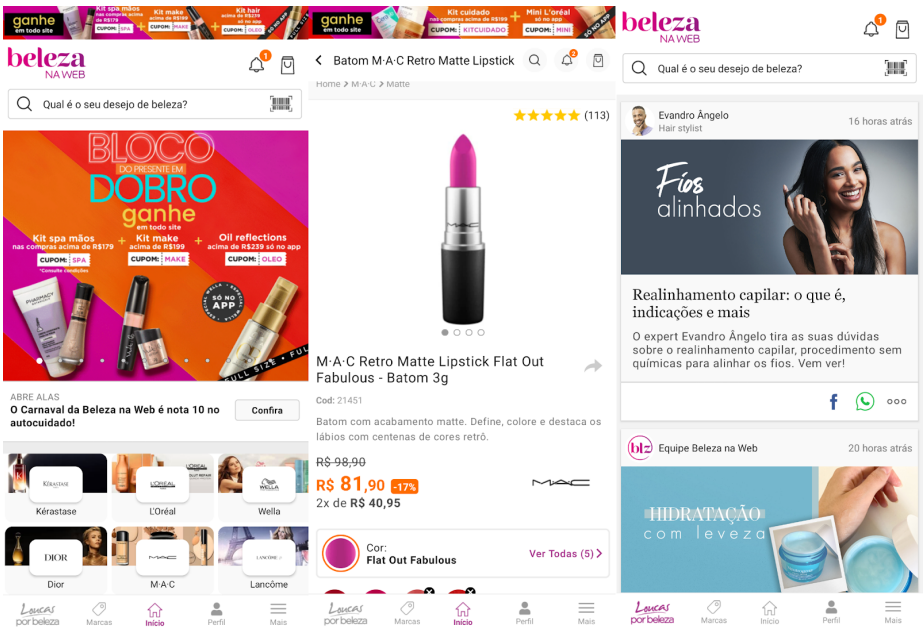
Figura 7 - Telas do Sephora



Fonte: Capturas de tela do aplicativo Sephora

- Beleza na Web: Perfume|Cabelo: aplicativo de uma das maiores lojas de e-commerce de cosméticos no Brasil, que foi fundada em 2008 e que desde o começo da empresa utilizou do gatilho mental de autoridade para convencer consumidores a comprar os produtos online, sem tocá-los, produzindo conteúdos de beleza sobre seus produtos com cabeleireiros famosos. Em 2019, a Beleza na Web foi comprada pelo Grupo Boticário. (RIVIERA, 2019). O aplicativo possui mais de 1 milhão de downloads na Google Play Store. Link do aplicativo na Google Play Store: https://play.google.com/store/apps/details?id=br.com.belezanaweb.android.

Figura 8 - Telas do Beleza na Web



Fonte: Capturas de tela do aplicativo Beleza na Web

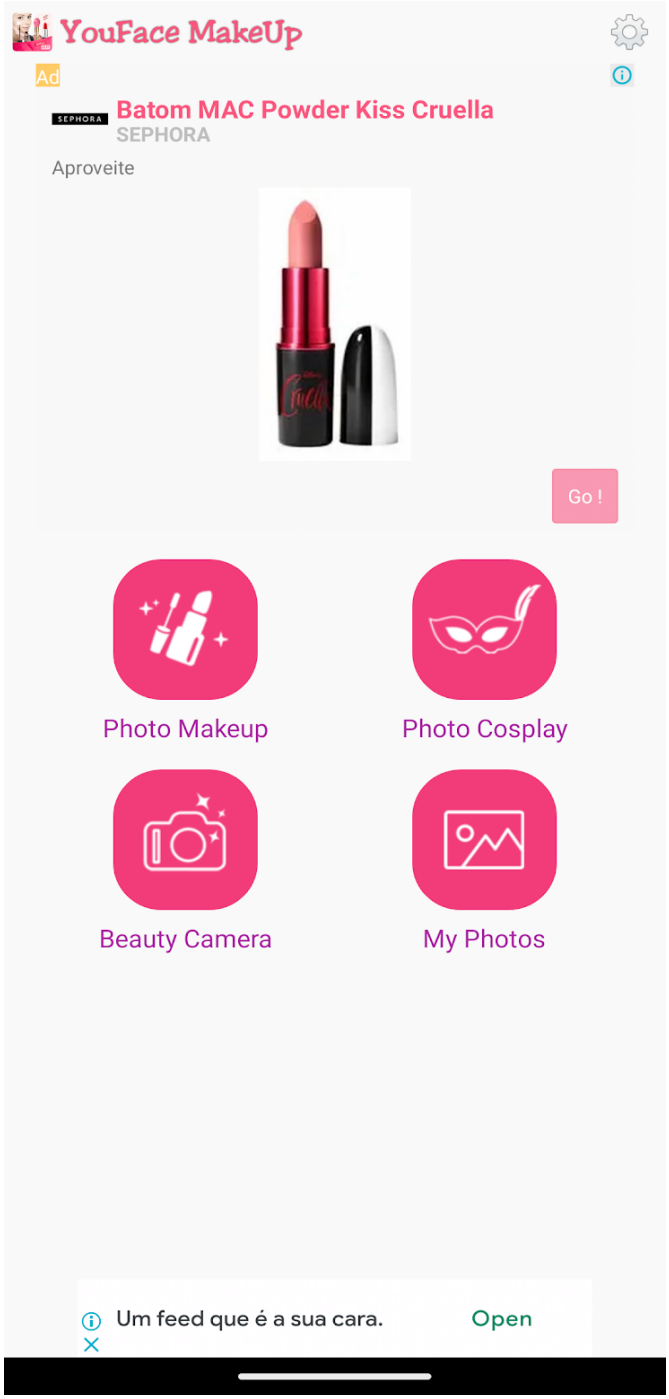
- Amazon: referência mundial no e-commerce, a Amazon é uma das marcas mais valiosas do mundo. (PEZZOTTI, 2021). O aplicativo possui mais de 500 milhões de downloads na Google Play Store e apesar de não ser especializada em maquiagem, possui muitos produtos do setor. Ao contrário de versões internacionais, dos Estados Unidos e do Japão (ENGLAND, 2019; WILLIAMS, 2019) a versão do aplicativo disponibilizada no Brasil não possui suporte para *try-on* com realidade aumentada. Link do aplicativo na Google Play Store: https://play.google.com/store/apps/details?id=com.amazon.mShop.android.shopping.

3.2 Aplicativos de *try-on* de maquiagem

Os aplicativos desta categoria utilizam realidade aumentada para criar filtros e aplicar maquiagem em fotos e na câmera em tempo real. Eles não possuem produtos reais nem e-commerce. Dentre os aplicativos analisados, apenas o aplicativo “YouFace Makeup - Makeover Studio” entrou nesta categoria. Nele, o usuário consegue utilizar maquiagens genéricas em suas fotos, além de usar filtros em tempo real. A falta de fidelidade dos filtros com o mundo real e a quantidade excessiva de anúncios são pontos negativos do aplicativo. Ele possui mais de 10

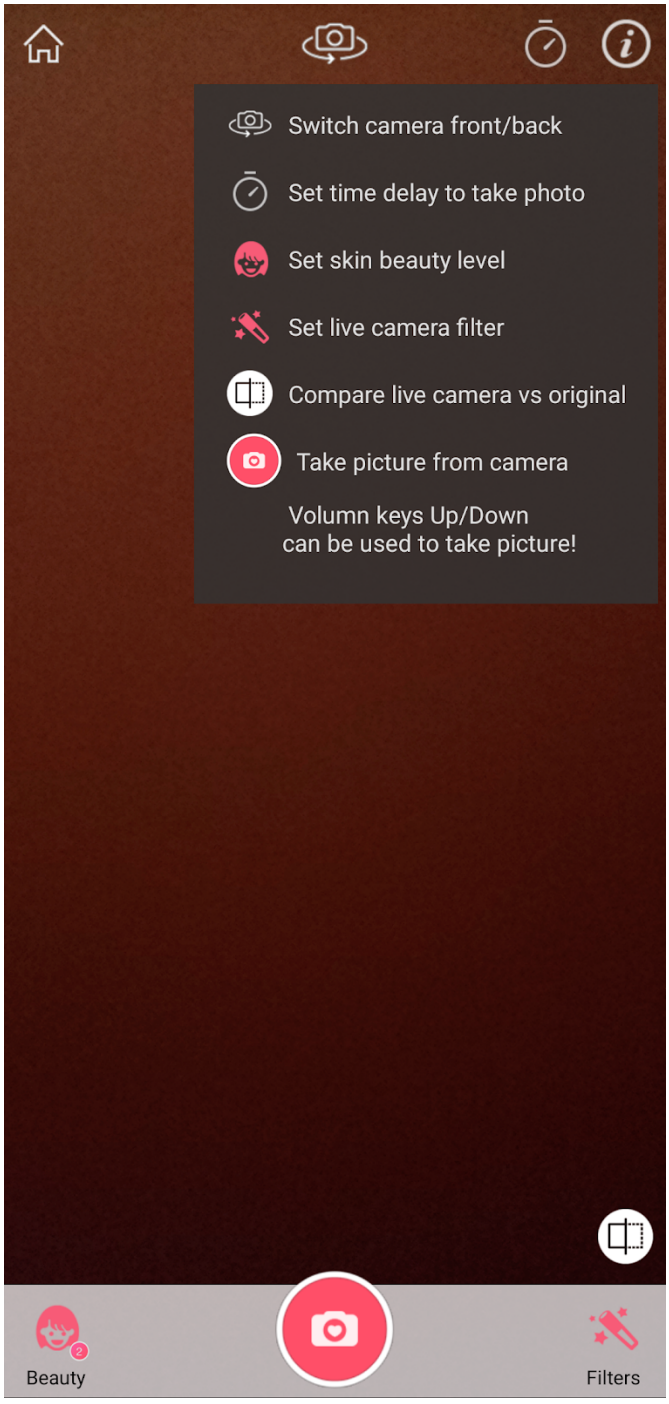
milhões de downloads na Google Play Store e pode ser acessado através deste link: https://play.google.com/store/apps/details?id=com.yubitu.android.YouFace.

Figura 9 - Tela inicial do YouFace Makeup



Fonte: Captura de tela do aplicativo YouFace Makeup - Makeover Studio

Figura 10 - Tela de *try-on* do YouFace Makeup



Fonte: Captura de tela do aplicativo YouFace Makeup - Makeover Studio

3.3 Aplicativos de e-commerce com *try-on* de maquiagem

Nesta categoria, os aplicativos possuem foco no *e-commerce*, na navegação na loja virtual, nos filtros dos produtos, e possuem a opção para visualizar o produto com realidade aumentada. É comum que nos aplicativos desta categoria a realidade aumentada seja um diferencial, não estando presente em muitos produtos. Serão

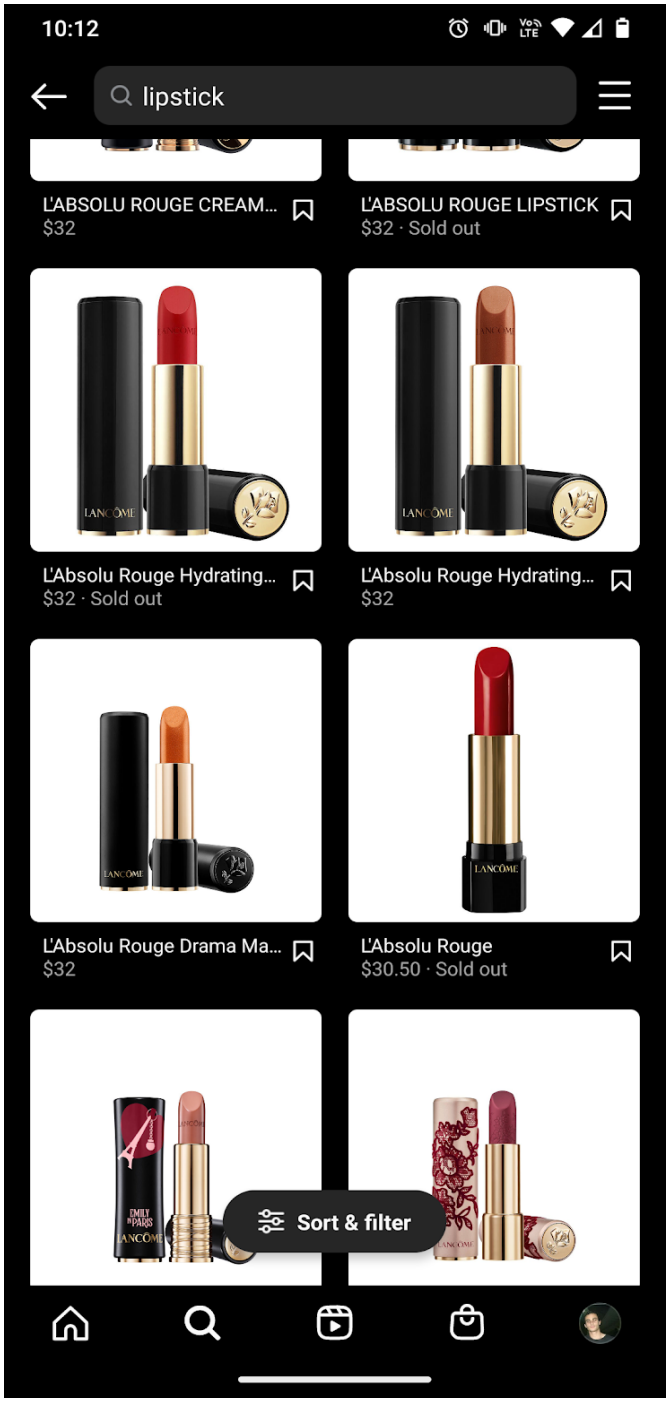
detalhados individualmente os aplicativos desta categoria e como eles utilizam a RA.

3.3.1 Instagram

A popular rede social de compartilhamento de fotos e vídeos possui mais de 1 bilhão de downloads na Google Play Store. Ao longo dos anos, o Instagram vem adicionando funcionalidades relacionadas a *e-commerce*: em 2016 foi introduzida, de forma experimental para um grupo de lojas, a funcionalidade de adicionar *tags* que adicionam o ícone “Toca para Ver” no canto inferior da foto que quando selecionada apresentava etiquetas dos produtos da foto com seus respectivos preços. Essas etiquetas, quando selecionadas, apresentavam mais informações sobre o produto e possuía o botão “Comprar Agora” que direcionava o usuário à página do produto no site da empresa (INSTAGRAM, 2016), em 2017 essa funcionalidade foi anunciada e aberta para todos os usuários (INSTAGRAM, 2017), em 2018 a funcionalidade de *tags* chegou também aos *stories* (INSTAGRAM, 2018), e, em 2020, foi introduzido o conceito de *Instagram Shop*, que são vitrines virtuais, nas quais os donos das páginas conseguem colocar seus produtos à mostra, com suas informações e preços. (INSTAGRAM, 2020) As vitrines virtuais possuem o que o Instagram chama de Páginas de detalhes dos produtos (PDP), onde são descritas as informações do produto e que possui o botão que direciona o usuário para o site da loja. (INSTAGRAM, 2022).

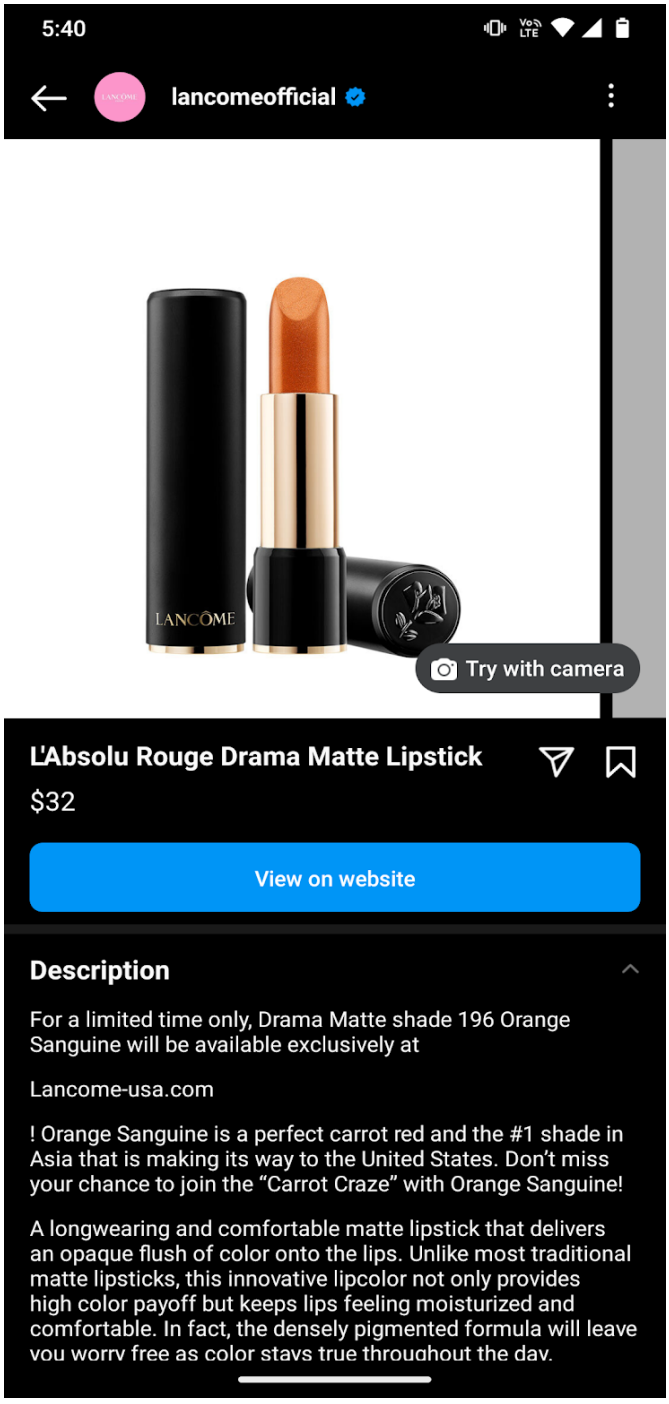
O aplicativo dá suporte à realidade aumentada, assim algumas empresas já possuem produtos com *try-on*. Na lista de produtos na vitrine, os produtos com *try-on* são apresentados com uma faixa com a mensagem “Experimentar”. Ao abrir a PDP, aparece a opção “*Try with camera*” (ela possui esse texto mesmo na versão em português do aplicativo), que leva o usuário para a tela de visualização. Na tela de visualização o usuário se vê utilizando o produto através da utilização de RA e, além da visualização normal do produto, o usuário pode: mudar a intensidade de aplicação do produto, ir ao site da loja, tirar fotos e compartilhá-las.

Figura 11 - Tela vitrine virtual de uma loja de cosméticos do Instagram



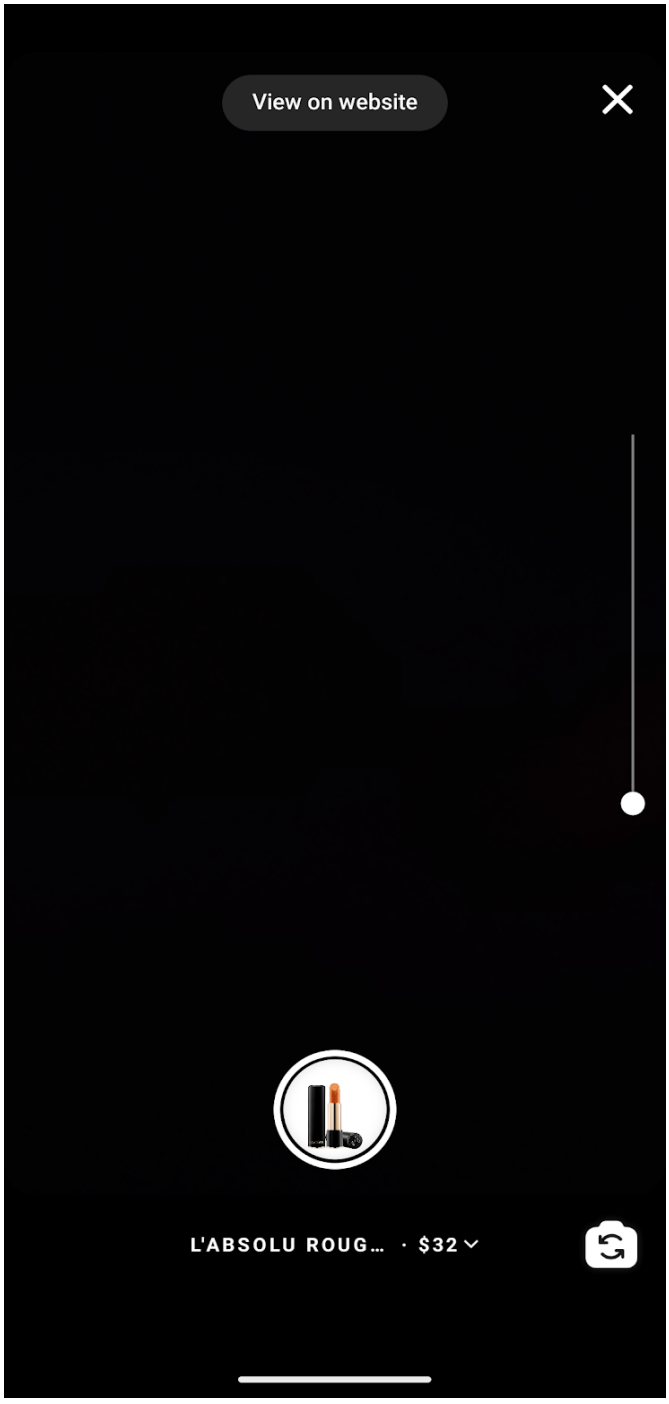
Fonte: Captura de tela do aplicativo Instagram

Figura 12 - Tela PDP do Instagram



Fonte: Captura de tela do aplicativo Instagram

Figura 13 - Tela de *try-on* do Instagram



Fonte: Captura de tela do aplicativo Instagram

No momento, apenas uma parcela muito pequena das empresas utilizam da funcionalidade de *try-on*, assim poucos produtos podem ser experimentados através do Instagram. É importante ressaltar que para testar os produtos os usuários precisam ir até a vitrine virtual da marca, eles não são apresentados nos filtros na tela de tirar foto. Desta forma, o usuário precisa primeiro conhecer a marca e o produto para experimentá-lo. Outro ponto negativo é que para mudar de produto o

usuário precisa sair da tela de visualização e ir para a PDP do outro produto que deseja experimentar, tornando a navegação mais lenta e dificultando comparar produtos, por exemplo para decidir qual cor de batom comprar.

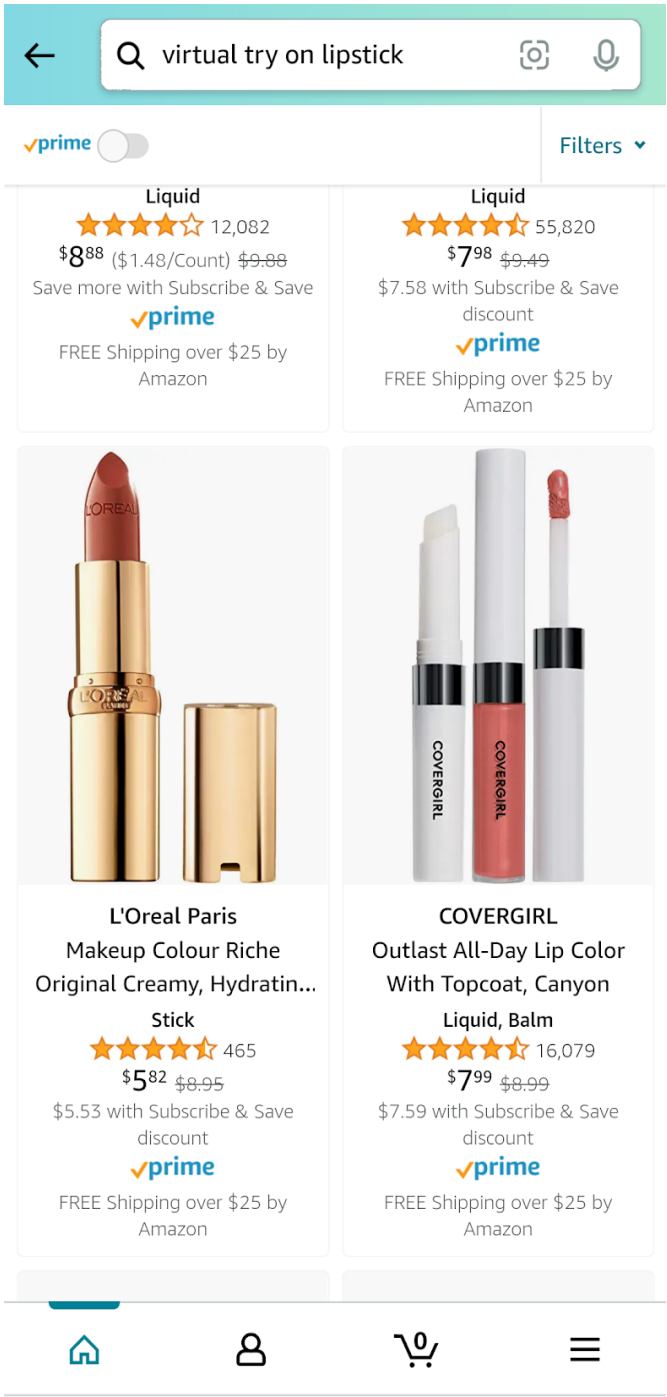
Link do aplicativo na Google Play Store: https://play.google.com/store/apps/details?id=com.instagram.android.

3.3.2 Amazon (versão estadunidense)

Como citado anteriormente, a versão estadunidense do aplicativo da Amazon possui *try-on* dos produtos com realidade aumentada. Para instalar o aplicativo, foi utilizada uma VPN para acessar a Google Play Store dos Estados Unidos.

Dentro do aplicativo, na listagem de produtos, não há nenhum indicativo de que o produto tem ou não *try-on* virtual, apenas na página do produto que aparece o botão "*try-on"*. Apesar disso, os produtos que possuem *try-on* podem ser filtrados, utilizando na busca as palavras-chave "*virtual try-on*". Na tela de *try-on*, é possível mudar a intensidade de aplicação do produto, adicionar o produto ao carrinho de compras, navegar entre outras cores do mesmo produto e utilizar fotos de modelos ao invés da câmera para visualizar o produto. Uma quantidade considerável de produtos estão disponíveis para *try-on* virtual. O aplicativo pode ser acessado, através da Google Play Store dos Estados Unidos, utilizando o link: https://play.google.com/store/apps/details?id=com.amazon.mShop.android.shopping.

Figura 14 - Tela de listagem de produtos do Amazon

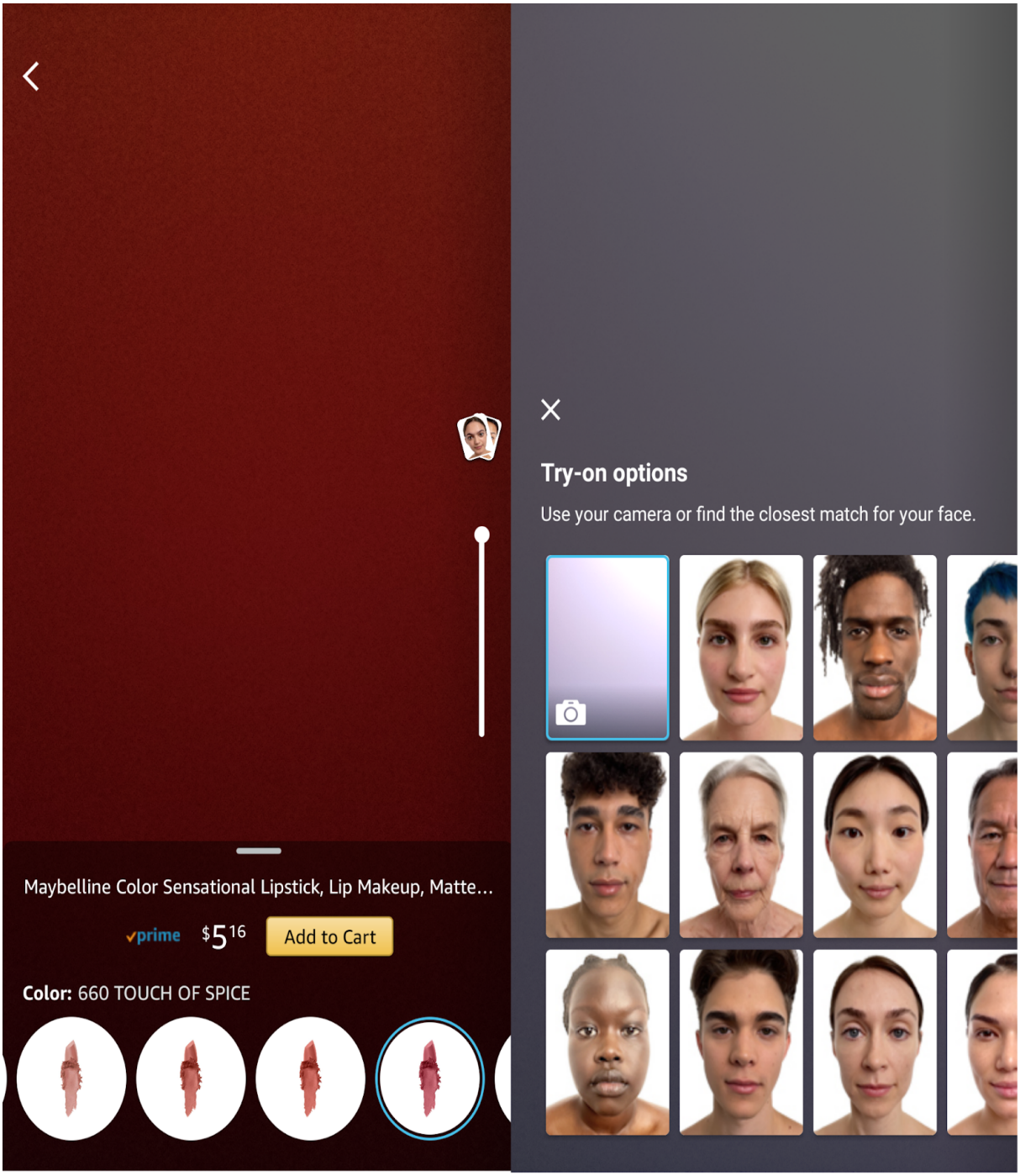


Fonte: Captura de tela do aplicativo Amazon

Figura 15 - Tela PDP do Amazon

Fonte: Captura de tela do aplicativo Amazon

Figura 16 - Telas *try-on* do Amazon



Fonte: Capturas de tela do aplicativo Amazon

3.3.3 Sephora - Beauty Shopping

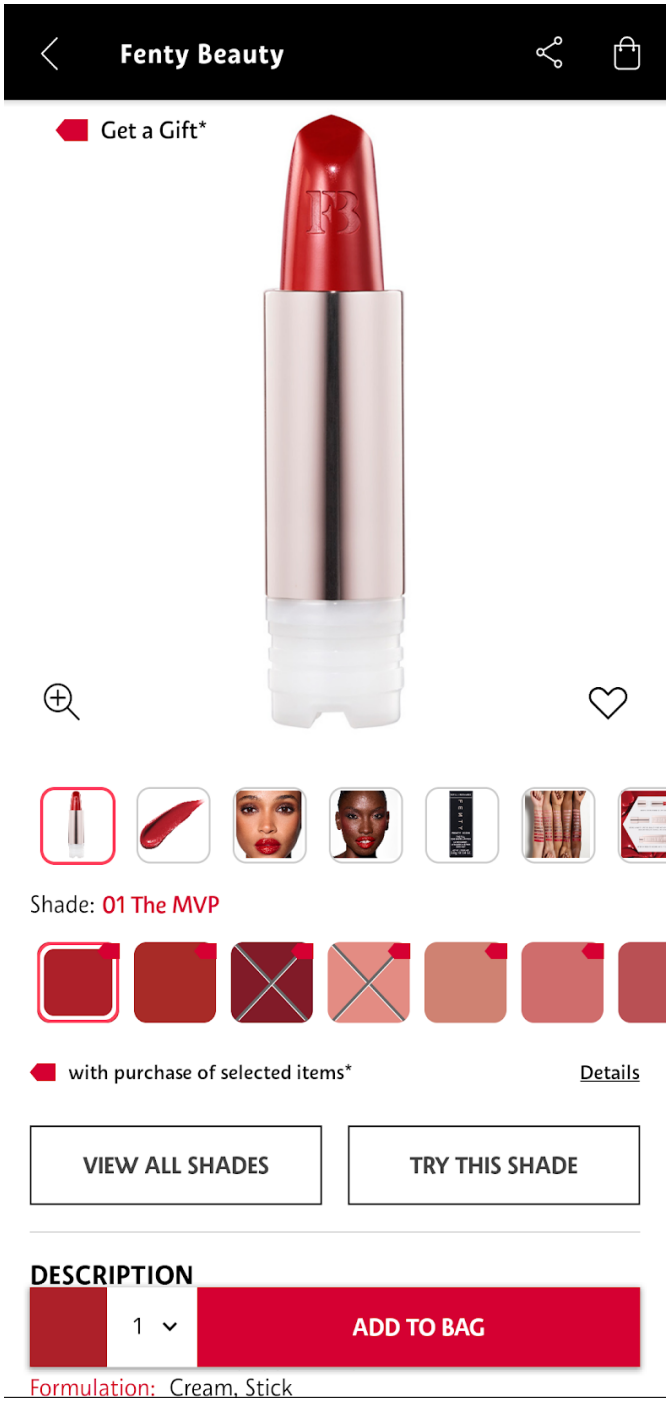
Como citado anteriormente, a Sephora possui aplicativos diferentes para cada uma das regiões em que atua. A versão analisada é feita para a região Ásia-Pacífico, sendo disponível nos seguintes lugares: Singapura, Malásia, Austrália, Nova Zelândia, Filipinas, Hong Kong, Tailândia e Indonésia. Para instalar este

aplicativo, foi utilizada uma VPN para acessar a Google Play Store de Singapura.

O aplicativo se apresenta como um aplicativo comum de e-commerce, porém, ao selecionar um produto, há a opção para experimentar o produto que leva à tela de visualização, que neste aplicativo é chamada de "*Visual Artist*". No primeiro acesso à essa tela, o tempo de processamento é um pouco grande, tendo uma tela de carregar por cerca de 3 minutos, porém os próximos acessos ao *Visual Artist* são instantâneos. A tela do *Visual Artist* possui no topo o produto que está sendo experimentado, seu preço, botões para adicioná-lo ao carrinho de compras e aos favoritos, e um botão que lista todos os produtos que estão sendo experimentados. A parte inferior possui a navegação, com ela o usuário consegue facilmente mudar de cor do produto, e experimentar outros produtos, sem precisar sair do *Visual Artist*. No *Visual Artist* é possível experimentar múltiplas maquiagens ao mesmo tempo, formando um *look*. Existe também a opção "Surprise me", que sugere uma maquiagem ao usuário. Ao tirar uma foto, ela é adicionada aos *looks* do usuário, que pode montar múltiplos *looks* e vê-los lado a lado, ajudando o usuário a decidir quais produtos comprar.

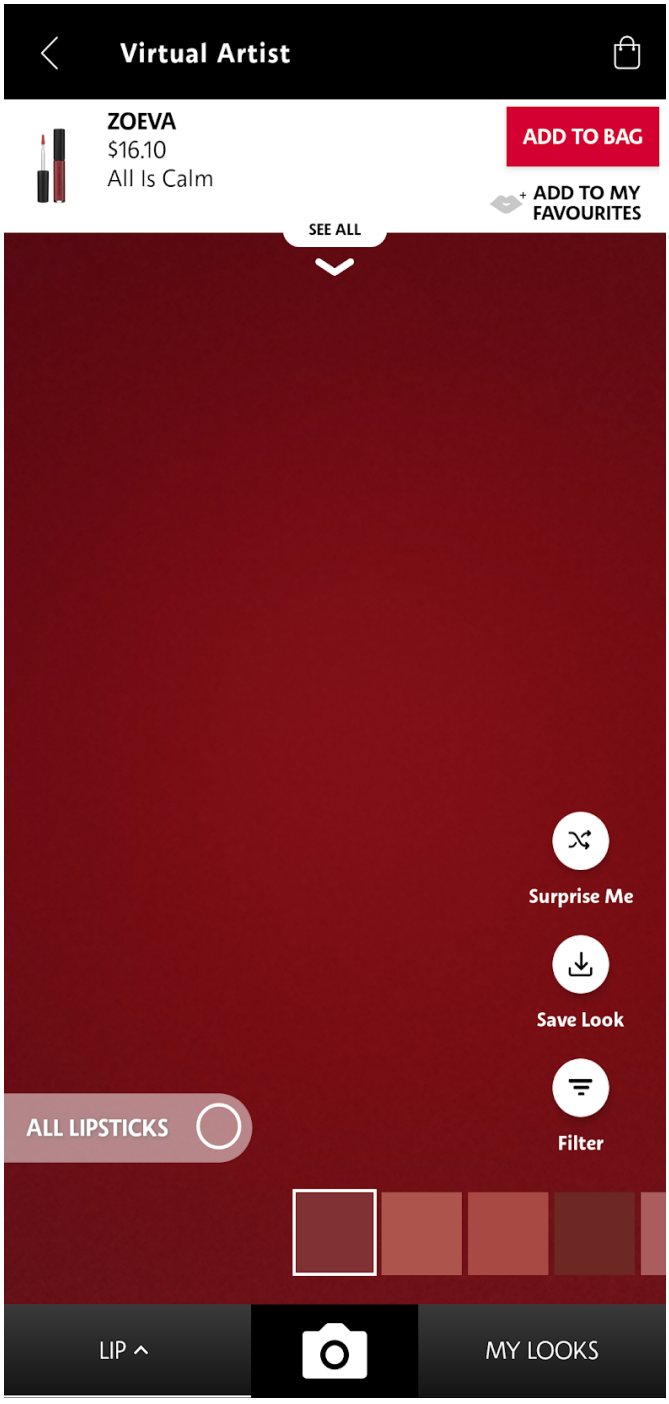
Link do aplicativo na Google Play Store (indisponível no Brasil): https://play.google.com/store/apps/details?id=com.sephora.digital.

Figura 17 - Tela PDP do Sephora



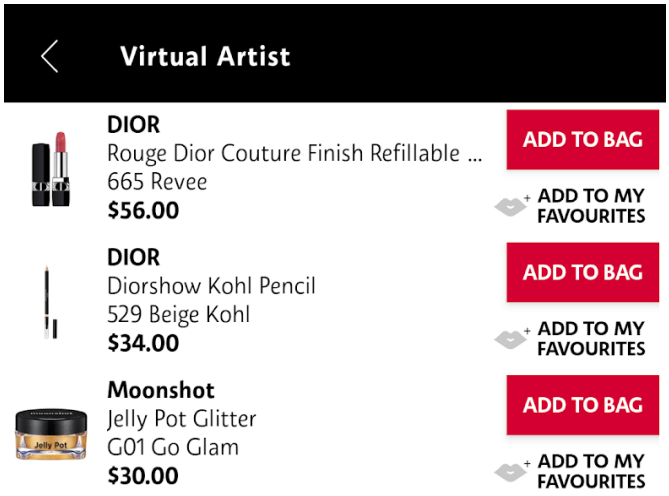
Fonte: Captura de tela do aplicativo Sephora

Figura 18 - Tela de *try-on* do Sephora (*Visual Artist*)



Fonte: Captura de tela do aplicativo Sephora

Figura 19 - Tela visualização dos produtos utilizados no *Visual Artist* do Sephora



Fonte: Captura de tela do aplicativo Sephora

3.4 Aplicativos de *try-on* de maquiagem voltados ao e-commerce

Ao contrário dos aplicativos da categoria anterior, os aplicativos desta categoria têm como foco a câmera com realidade aumentada, onde os usuários experimentam maquiagens e filtros. Eles possuem produtos reais, e mostram ao usuário como comprá-los. Serão detalhados individualmente os aplicativos desta

categoria e como eles utilizam a RA.

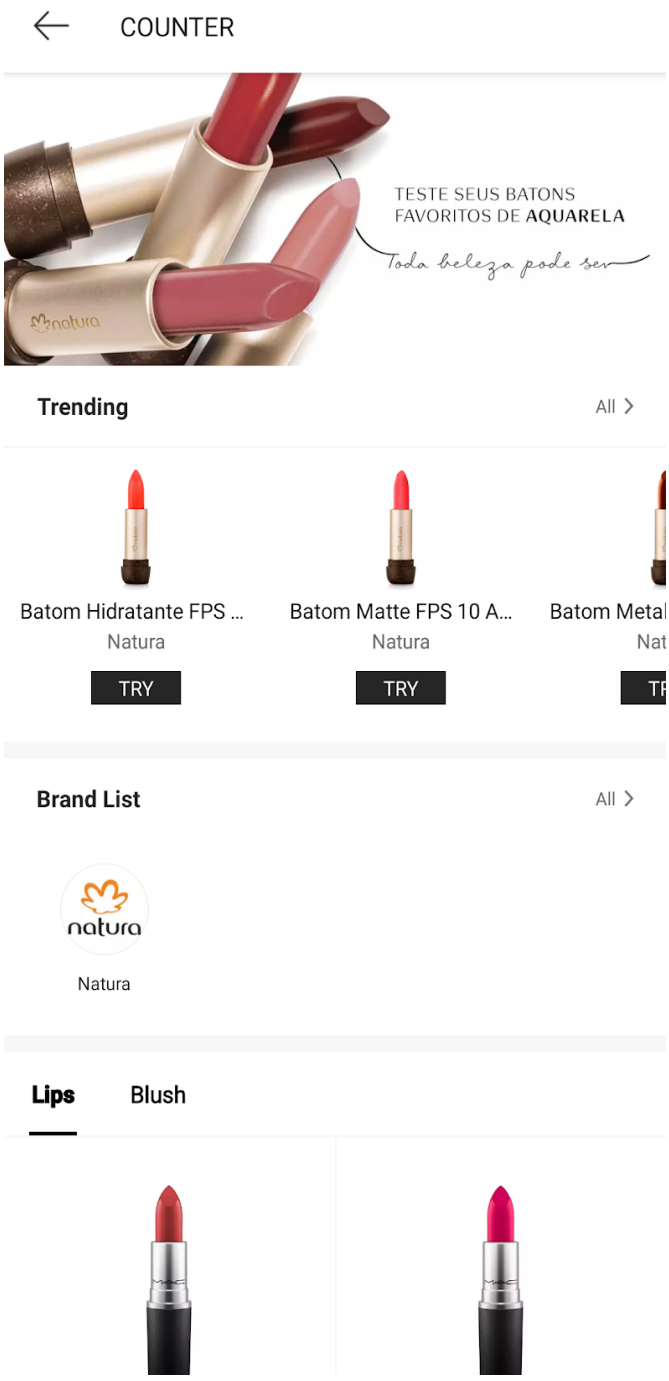
3.4.1 MakeupPlus - Virtual Makeup

Lançado em 2015 pela empresa chinesa Meitu, o MakeupPlus inicialmente servia para usuários adicionar maquiagem e filtros a suas fotos. Posteriormente foi adicionada a funcionalidade de *try-on* com a câmera em tempo real. A Meitu se descreve como “Uma empresa de tecnologia orientada a inteligência artificial que carrega a beleza como seu principal ideal, ajudando seus usuários a se tornarem mais bonitos com produtos de imagem e serviços de gerenciamento de beleza e auxiliando na transformação digital do setor de beleza fornecendo software como serviço”. Os produtos da Meitu visam atingir tanto os usuários quanto a indústria de beleza, oferecendo, aos usuários, *try-on* de maquiagem em alta definição e , à indústria, marketing, pois seus produtos são apresentados para o usuário, e dados do comportamento dos usuários em relação aos produtos. (MEITU, 2022).

Na tela inicial do aplicativo, existem 4 botões: “Selfie - Live makeup”, “Counter”, “Touch-up” e “Trending”. O botão de “Trending” leva para uma página de anúncios de aplicativos de celular. O botão de “Selfie - Live makeup” abre a câmera e o usuário pode experimentar filtros pré definidos ou utilizar maquiagens genéricas. Ao selecionar o botão “Touch-up”, o usuário consegue utilizar esses mesmos filtros e maquiagens genéricas em fotos. Ao selecionar o botão “Counter”, é apresentada uma lista de produtos reais pelos quais o usuário pode experimentar. Esta é a função relevante a este trabalho. Ao selecionar um produto é aberta a câmera e na tela de *try-on* o usuário pode: mudar a tonalidade do produto, mudar a intensidade de aplicação do produto e visualizar detalhes do produto. Ao selecionar visualizar o produto, é aberta uma página de visualização de forma tradicional e nela há um botão de comprar o produto, que direciona o usuário para a página de e-commerce da marca do produto. Alguns links, porém, estão desatualizados, direcionando o usuário para páginas de erro.

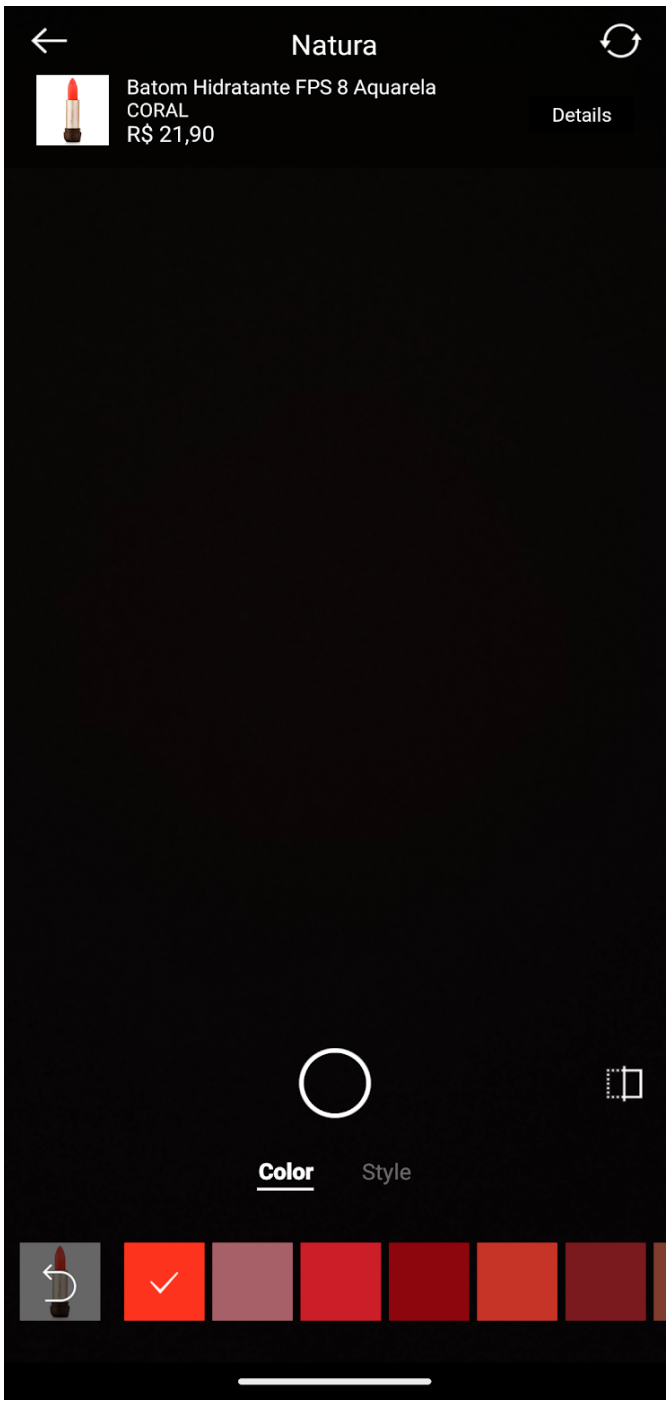
O aplicativo possui mais de 50 milhões de downloads na Google Play Store e pode ser acessado através do link: https://play.google.com/store/apps/details?id=com.meitu.makeup&hl=pt_BR&gl=US.

Figura 20 - Tela de listagem dos produtos (*counter)* do MakeupPlus



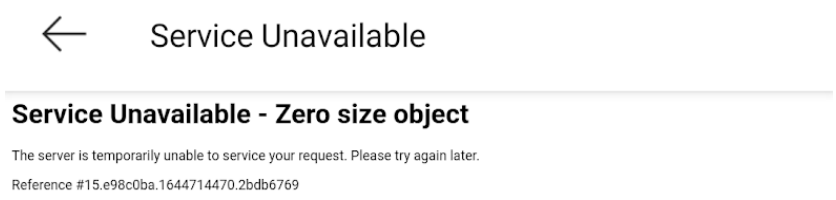
Fonte: Captura de tela do aplicativo MakeupPlus - Virtual Makeup

Figura 21 - Tela de *try-on* do MakeupPlus



Fonte: Captura de tela do aplicativo MakeupPlus - Virtual Makeup

Figura 22 - Mensagem da tela de redirecionamento do MakeupPlus



Fonte: Captura de tela do aplicativo MakeupPlus - Virtual Makeup

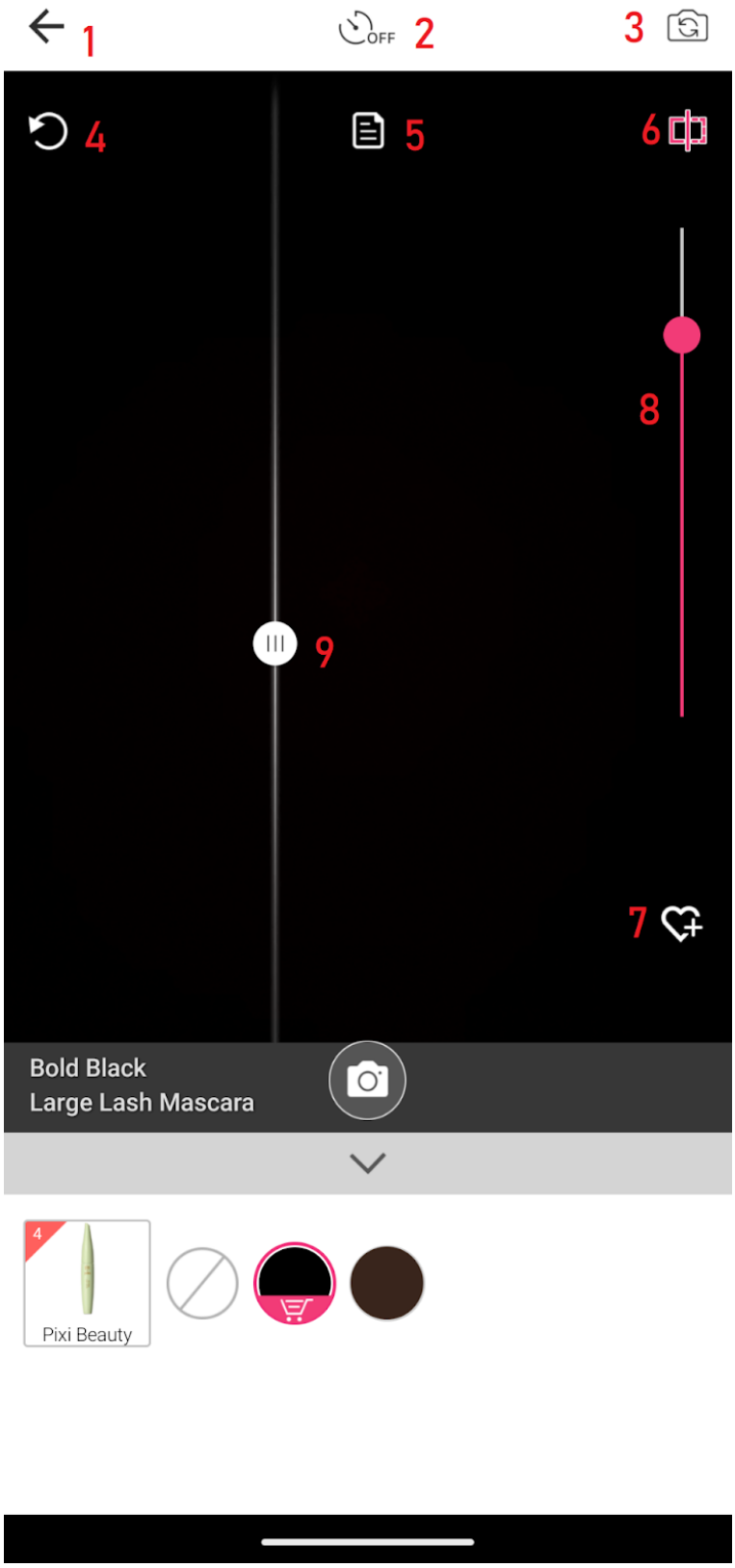
3.4.2 YouCam Makeup - Selfie Editor

Produzido pela Perfect Corp., o YouCam Makeup é um dos aplicativos de edição de *selfies* mais populares do mundo, com mais de 100 milhões de downloads na Google Play Store. A Perfect Corp. se descreve como “a principal fornecedora de soluções de negócios como *Service as a Software* (SaaS) de tecnologia de moda e beleza de realidade aumentada e inteligência artificial, dedicada a transformar a experiência de compra do consumidor por meio de experiências perfeitas e omnicanal”. (PERFECT CORP, 2022d). Além de seus aplicativos, a Perfect Corp. também fornece soluções para empresas, levando a RA para os pequenos e médios negócios. (PERFECT CORP, 2022a). Essas soluções possuem casos de sucesso, como o da empresa nova-iorquina e.l.f Cosmetics, que teve um aumento de 200% na taxa de conversão (PERFECT CORP, 2022b) e o da californiana Benefit Cosmetics, que teve um aumento de 101% no tempo de sessão dos usuários e de 20% na quantidade de vezes que itens foram adicionados ao carrinho de compras. (PERFECT CORP, 2022a).

O aplicativo possui uma variedade de funcionalidades, como avaliar a qualidade da pele, envelhecimento da pessoa em uma foto, aplicar maquiagem em fotos e em tempo real com a câmera, aplicar filtros e editar fotos, e funciona também como uma rede social, onde usuários e criadores de conteúdo compartilham dicas de maquiagem e *looks*. A funcionalidade que entra no escopo deste trabalho é a de aplicar maquiagem em fotos e, principalmente, em tempo real com a câmera. A tela da visualização com RA (Figura 23) possui em seu topo uma barra, com botões para

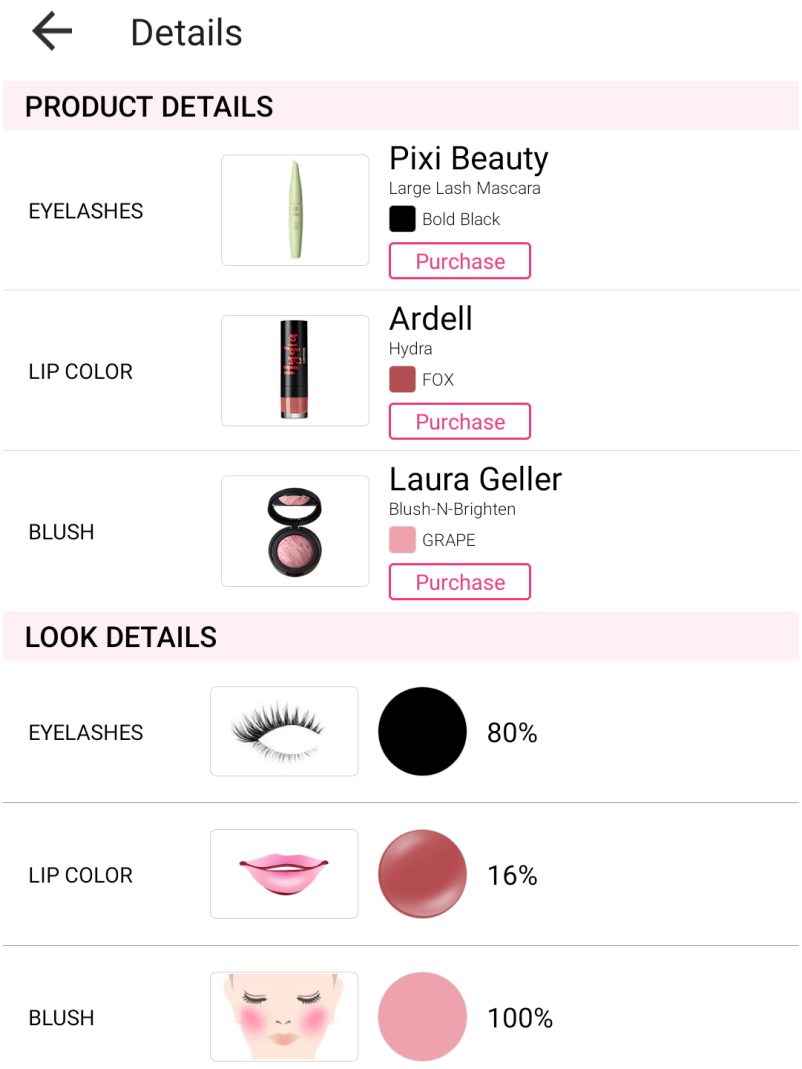
(1) voltar ao menu principal , (2) temporizador de foto e (3) mudar de câmera. Dentro da parte principal, há botões de (4) remover todas as maquiagens e filtros aplicados, (5) visualizar os produtos e filtros utilizados, (6) habilitar o *slider* de comparação entre a imagem real e a que possui RA e (7) adicionar aos favoritos; e *sliders* para (8) mudar a intensidade da aplicação do produto e (9) fazer a comparação entre a imagem normal da câmera e a que os filtros são aplicados. Na parte inferior, é possível navegar pelos produtos e, ao selecionar um produto, aparece o ícone de um carrinho de compras que, ao ser clicado, direciona o usuário para a página de e-commerce da marca do produto. O aplicativo possui uma grande variedade de tipos de produtos e marcas, além de funcionalidades úteis e boa usabilidade. O aplicativo pode ser acessado através do link: https://play.google.com/store/apps/details?id=com.cyberlink.youcammakeup&hl=pt_BR&gl=US.

Figura 23 - Tela de Câmera do YouCam Makeup - Selfie Editor



Fonte: Captura de tela do Autor modificada

Figura 24 - Tela Visualização dos produtos utilizados na Câmera YouCam Makeup - Selfie Editor



Fonte: Captura de Tela do Autor

3.5 Comparativo

A seguir é apresentada uma tabela comparativa entre os aplicativos analisados:

Tabela 1 - Comparativo entre os aplicativos analisados

Aplicativo	**Realidade Aumentada**	**Navegação entre produtos na tela de RA**	**E-commerce**	**App Nacional**
Época Cosméticos e Maquiagens	Não	Não aplicável	Sim	Sim

Sephora	Não	Não aplicável	Sim	Sim
Beleza na Web: Perfume | Cabelo	Não	Não aplicável	Sim	Sim
Amazon (versão do Brasil)	Não	Não aplicável	Sim	Versão Nacional
YouFace Makeup - Makeover Studio	Sim	Sim	Não	Não
Instagram	Sim, em poucos produtos	Não	Direciona para o website da loja	Versão Nacional
Amazon (versão estadunidense)	Sim	Entre cores do mesmo produto	Sim	Não
Sephora - Beauty Shopping	Sim	Sim	Sim	Não, e não está disponível no Brasil
MakeupPlus - Virtual Makeup	Sim	Sim	Direciona para o website da loja	Não
YouCam Makeup - Selfie Editor	Sim	Sim	Direciona para o website da loja	Não

Fonte: Autor

A Tabela 1 faz a comparação entre os aplicativos analisados. Ela possui 5 colunas: “Aplicativo”, que é o nome do aplicativo analisado, “Realidade Aumentada”, que indica se o aplicativo utiliza realidade aumentada na apresentação do produto ao usuário, “Navegação entre produtos na tela de RA”, que indica se é possível experimentar diferentes produtos na tela de visualização com RA, “E-commerce”, que indica se o aplicativo possui o objetivo de vender os produtos nele apresentados, e “App Nacional”, que indica se o aplicativo foi desenvolvido no Brasil ou possui uma versão brasileira.

Ao analisar a tabela, é possível encontrar algumas relações. A mais evidente é que aplicativos focados no e-commerce geralmente não possuem realidade

aumentada. Outra é que aplicativos com o foco na realidade aumentada para o *try-on* de maquiagem direcionam os usuários para o *site* do produto, ao invés de ter o processo de e-commerce integrado. Apenas os aplicativos da Amazon (versão estadunidense) e Sephora - Beauty Shopping, possuem tanto o *try-on* quanto o e-commerce no próprio aplicativo, e ambos não estão disponíveis no Brasil. Outra relação é que os aplicativos nacionais não possuem realidade aumentada, nem a versão nacional do aplicativo da Amazon. Um fator que pode ter sido determinante para isso é a baixa quantidade de pessoas com dispositivos com capacidade para utilizar a tecnologia. Esse cenário, porém, vem mudando e é provável que estes aplicativos venham a aderir a realidade aumentada para exibir seus produtos nos próximos anos.

4. PROPOSTA

Após realizar os estudos sobre realidade aumentada e seus impactos no processo de decisão de compra do consumidor e de analisar aplicativos populares relacionados à maquiagem, foi elaborada a solução proposta para este trabalho. Foram consideradas para a proposta, as funcionalidades encontradas pelos aplicativos analisados que o autor considerou mais úteis e impactantes. O sistema é um aplicativo móvel para o sistema operacional Android e se chama MaquiAR. O MaquiAR permitirá ao usuário experimentar diferentes tipos de maquiagens, através de filtros. Ao utilizar os filtros o usuário pode tirar fotos, compartilhar em redes sociais, salvar maquiagens em seus favoritos para vê-las depois. O usuário tem acesso a informações das maquiagens que estão sendo utilizadas no filtro que ele está usando, além de informações de como adquiri-las. O usuário pode, também, navegar pela lista de produtos disponíveis no aplicativo, além de visualizar seus favoritos.

A seguir é apresentado o posicionamento do MaquiAR na tabela utilizada para o comparativo dos aplicativos analisados anteriormente:

Tabela 2 - Posicionamento do MaquiAR na tabela de comparativo

Aplicativo	**Realidade Aumentada**	**Navegação entre produtos na tela de RA**	**E-commerce**	**App Nacional**
MaquiAR	Sim	Sim	Direciona para o website da loja	Sim

Fonte: Autor

O MaquiAR entra na categoria de aplicativos de *try-on* voltados ao e-commerce, pois o foco do aplicativo é nos filtros de maquiagem através da realidade aumentada e tem a visão de que ao experimentar maquiagens o usuário pode desejar comprá-las, então o aplicativo fornecerá ao usuário onde comprá-las. Aplicativos que direcionam para as páginas das lojas possuem, em sua maioria, parcerias com as lojas, de tal forma que os aplicativos recebem das lojas ou uma porcentagem sobre as compras de clientes vindos dos links fornecidos por esses aplicativos ou um valor sobre cada cliente que acessou o link. As maquiagens do aplicativo serão genéricas e a funcionalidade de mostrar informações do produto e

onde comprá-lo avisará o usuário que estão sendo utilizadas maquiagens genéricas, mostrará produtos reais que possuem características semelhantes à maquiagem genérica e avisará o usuário que o nível de fidelidade da maquiagem apresentada pode ter pequenas diferenças para os produtos.

4.1 Requisitos

Os requisitos funcionais do sistema foram separados, de acordo com sua prioridade. Foram adotadas as denominações "essencial" e “desejável”. Os requisitos essenciais são imprescindíveis para o funcionamento do sistema, são obrigatórios para o sucesso do projeto; enquanto os requisitos desejáveis são requisitos que não comprometem as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis são requisitos que podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

4.1.1 Requisitos Funcionais Essenciais

- RFE1 - Visualizar a lista de produtos
- RFE2 - Visualizar os produtos na forma tradicional, com texto e imagens
- RFE3 - Visualizar os produtos com Realidade Aumentada
- RFE4 - Mudar de produto que está sendo experimentado sem sair da tela de RA
- RFE5 - Adicionar, remover e visualizar os produtos favoritos
- RFE6 - Tirar e compartilhar fotos na visualização com RA
- RFE7 - Direcionar o usuário para a página web onde o produto é comercializado

4.1.2 Requisitos Funcionais Desejáveis

- RFD1 - Visualizar o histórico de produtos visualizados com RA
- RFD2 - Visualizar e mais de um produto com RA ao mesmo tempo
- RFD3 - Mudar a intensidade de aplicação do produto na visualização

com RA

- RFD4 - Possuir um *Slider* para comparação da imagem com e sem RA
- RFD5 - Possuir produtos reais

4.1.3 Requisitos Não Funcionais

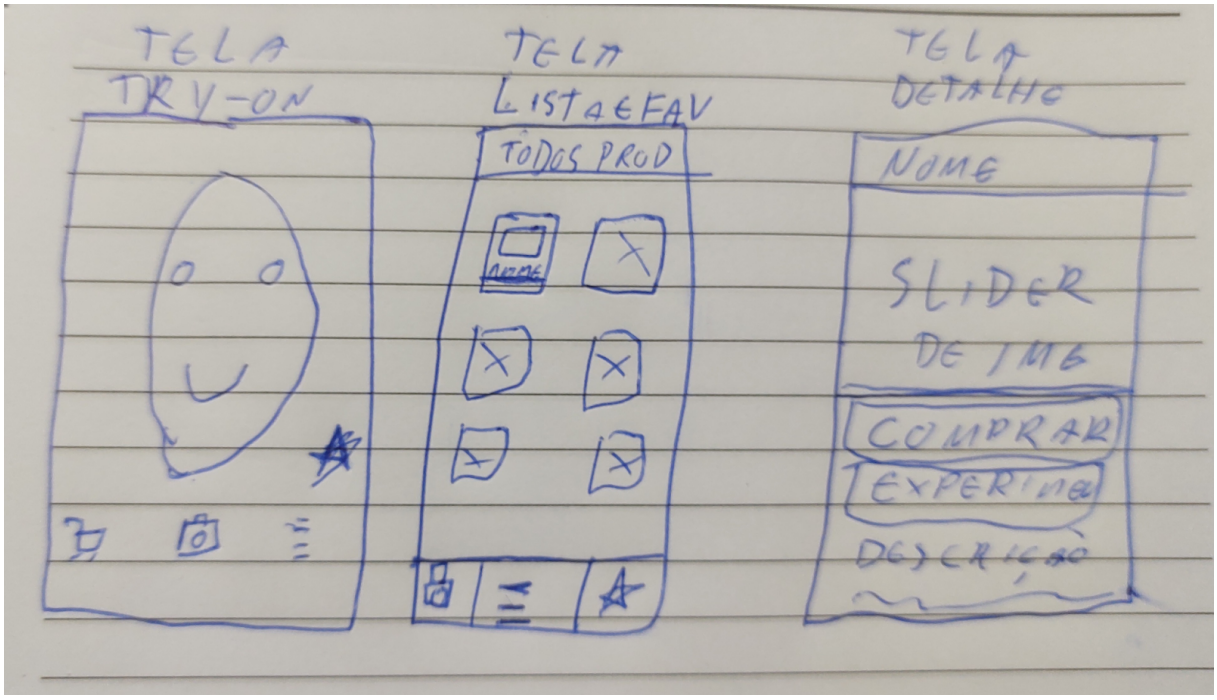
- RNF1 - Ser compatível com dispositivos Android
- RNF2 - Interface simples e fáceis de usar

5. DESENVOLVIMENTO

Após os requisitos terem sido levantados, começou o processo de desenvolvimento do aplicativo. Primeiramente, foi feito o design da interface do usuário. A ideia do aplicativo precisa ser traduzida numa interface. Não é suficiente dizer que o aplicativo faz X, Y e Z; é preciso que o usuário do aplicativo veja, em cada tela, que o aplicativo faz X, Y e Z. (ANTHONY, 2012)

Foram feitas *sketches* das interfaces, colocando no papel a ideia do aplicativo. *Sketches* são desenhos, muitas vezes feitos à mão, das telas do aplicativo e são muito úteis para o design de interfaces. *Sketching* é uma ferramenta muito poderosa no processo criativo, ela auxilia no entendimento da solução, na percepção de detalhes e até no descobrimento de novas funcionalidades. “Sua primeira ideia raramente é a melhor”. (LAMP, 2011). Ao fazer *sketches* fazemos mudanças na ideia da interface que custariam muito mais caso deixássemos para perceber esses detalhes apenas na implementação. O processo de *sketching* teve algumas iterações e as figuras a seguir são da última iteração, ou seja, o esboço final das interfaces:

Figura 25 - Esboços das interfaces do MaquiAR



Fonte: Autor

Após a definição das interfaces, foi iniciada a implementação do aplicativo. Foi utilizada para a implementação do aplicativo a linguagem Kotlin, ARCore versão 1.21, SDK mínima do Android versão 27 (Oreo 8.1.0) e SDK alvo do Android versão 30 (Android 11). Inicialmente, houve um foco na lógica básica do sistema, ou seja, nas classes que compõem o domínio do problema. Nessa etapa, foi criada a classe abstrata *Product*, que representa um produto, uma maquiagem genérica, e suas implementações, ou seja os tipos de maquiagem, batom, *blush* e sombra de olho. Na seção 5.1 será detalhada a arquitetura da aplicação.

Com as classes do domínio do problema implementadas, podemos começar a implementação das *Activities*. Aqui, separamos o desenvolvimento do aplicativo em duas partes: o desenvolvimento com RA, ou seja a tela de *try-on* das maquiagens, e o desenvolvimento das outras telas. Houve essa separação para o autor focar no entendimento do desenvolvimento de RA utilizando ARCore. Essa etapa foi de fundamental importância, pois é a parte principal do aplicativo, por isso foi empregado mais tempo para ela.

Finalizada a tela de *try-on*, foram feitas as texturas. As texturas foram feitas a partir da imagem “makeup.png” do projeto

Unity-ARFoundation-echoAR-Face-Makeup[1] , e editadas pelo autor de modo que elas ficassem brancas, para serem aplicadas as *shaders*, e com opacidade menor de forma que ficassem mais realistas. O conceito de *shaders* será explicado mais adiante neste trabalho.

Neste momento o aplicativo se encontra como um aplicativo de *try-on* de maquiagem. Começou, então, o desenvolvimento das outras telas: a que lista os produtos, a tela de descrição do produto - que possui o link para o site onde o usuário pode realizar a compra, e a tela de favoritos.

5.1 Arquitetura

Nesta seção serão descritas as classes do sistema, suas funções e como elas interagem entre si. A maior parte deste trabalho foi desenvolvida usando o padrão MVC (*Model-View-Controller*) com exceção da parte relacionada à realidade aumentada, que foi separada num diretório a parte e suas especificidades que serão comentadas posteriormente. Não foi utilizado nenhum banco de dados nem servidor, apesar de ser totalmente aplicável. O motivo se dá pelo foco do trabalho ser na realidade aumentada e no desenvolvimento mobile Android.

Na camada Model ficam as entidades do domínio: *Product*, *Blush, EyeShadow* e *Lipstick*. *Product* é uma classe abstrata que define as características de um produto para o sistema: id, nome, descrição, imagens, ícone do tipo de maquiagem, link do site que o usuário será direcionado, textura que será usada na tela de realidade aumentada, vetor com os valores correspondentes às cores que serão aplicadas à textura, e o nome da cor. As classes *Blush, EyeShadow* e *Lipstick* são implementações de *Product*. Seus construtores não possuem a textura e o ícone do tipo de maquiagem, pois estes são pré-definidos dentro da classe - cada tipo de maquiagem possui sua textura e seu ícone.

Na camada View ficam as entidades responsáveis em exibir os dados aos usuários. Ela é formada pelas *activities* e pelos *recyclers*:

- ARVisualizationActivity: responsável pela tela de *try-on.*

[1] https://github.com/VivianVKJ/Unity-ARFoundation-echoAR-Face-Makeup - Acesso em junho de 2022

- FavoriteActivity: responsável pela tela de favoritos.
- ProductDetailActivity: responsável pela tela PDP.
- ProductListActivity: responsável pela tela de listar produtos.
- ProductCardViewRecycler: responsável pelo *RecyclerView* de listar produtos, utilizado pelas *activities* de listar produtos e listar favoritos.
- ARVisualizationProductCardViewRecycler: responsável pelo *RecyclerView* de listar produtos na tela de *try-on*.

Na camada Controller ficam as entidades responsáveis em lidar com os dados e informar a View o que ela precisa exibir. Nela estão as classes:

- ProductLoader: responsável por carregar os produtos.
- FavoriteUtils: responsável pela lógica de favoritos.

As implementações destas classes serão detalhadas na seção 5.2.

Para a parte relacionada à realidade aumentada, foi utilizada a biblioteca *ARCore Augmented Faces wrapper without Sceneform*[2] . Ela é uma versão modificada do exemplo de *Augmented Faces* disponibilizado no github da ARCore SDK[3], que torna mais simples a adição de texturas e objetos a um rosto. Ele é formado pelos diretórios *Helpers* (classes de auxílio) e *Rendering* (classes responsáveis pela renderização dos objetos RA e *background*) e pelos arquivos:

- *AugmentedFaceFragment*: classe responsável pela renderização e rastreamento das *Augmented Faces.*
- *AugmentedFaceListener*: interface responsável pelo callback de adicionar e atualizar as *Augmented Faces.*
- *AugmentedFaceNode*: classe responsável pela renderização do rosto incluindo as texturas e os objetos 3D.
- *AugmentedFaceRenderer*: classe responsável por renderizar a textura.
- *FaceRegion*: classe responsável por renderizar os objetos 3D de uma parte do rosto.

---

[2] https://github.com/Kristina-Simakova/photobooth_codelab_start_project - Acesso em junho de 2022

[3] https://github.com/google-ar/arcore-android-sdk/tree/master/samples/augmented_faces_java Acesso em junho de 2022

O uso da biblioteca é simples, basta adicionar o *AugmentedFaceFragment* ao layout e implementar a interface *AugmentedFaceListener* na *activity* de visualização com RA, adicionando a textura referente à maquiagem e sua cor.

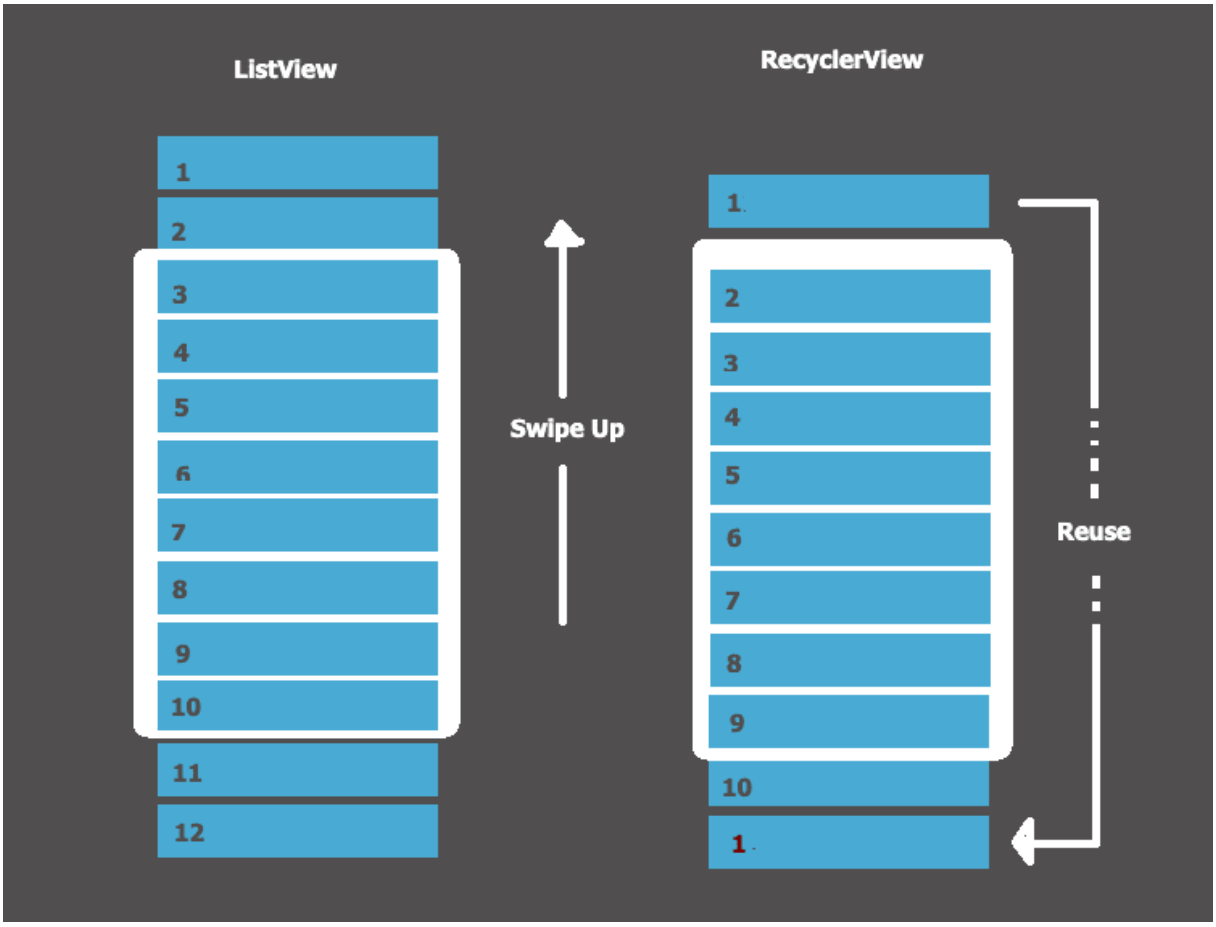
5.2 Implementação

Nesta seção serão apresentados especificidades da implementação do aplicativo, dificuldades encontradas no processo de desenvolvimento e decisões tomadas, além das interfaces. A seção é dividida por funcionalidade.

5.2.1 Visualizar lista de produtos

A funcionalidade de listar produtos está presente em duas telas do aplicativo: na tela inicial, que lista todos os produtos, e na tela de favoritos, que lista os produtos que o usuário adicionou a seus favoritos. Foi utilizado *RecyclerView* para implementar a lista. Ele é uma versão mais elaborada da *ListView*, que permite reutilizar um elemento já criado e repetir dentro da interface. Com *ListView*, caso tivéssemos uma lista com 100 elementos, teríamos que criar os 100 elementos, o que compromete muito a performance. Com *RecyclerView*, são criados apenas a quantidade de elementos que o usuário irá interagir, à medida que o usuário descer ou subir na tela, o componente identifica os elementos que não estão mais visíveis e os reutiliza colocando novos valores.

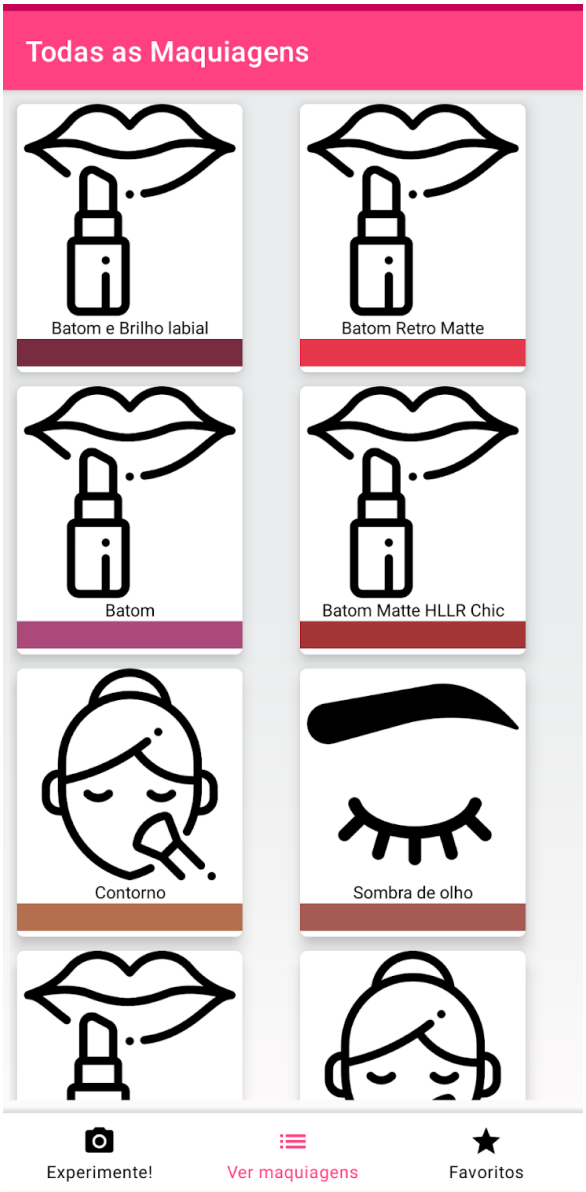
Figura 26 - Comparação entre *ListView* e *RecyclerView*



Fonte: BARRIENTOS, 2020

No caso deste trabalho, os elementos a serem exibidos são *cards* que representam os produtos. O *card* é composto pelo ícone referente ao tipo da maquiagem, o nome da maquiagem e a cor da maquiagem na base do card.

Figura 27 - Tela lista de produtos do MaquiAR



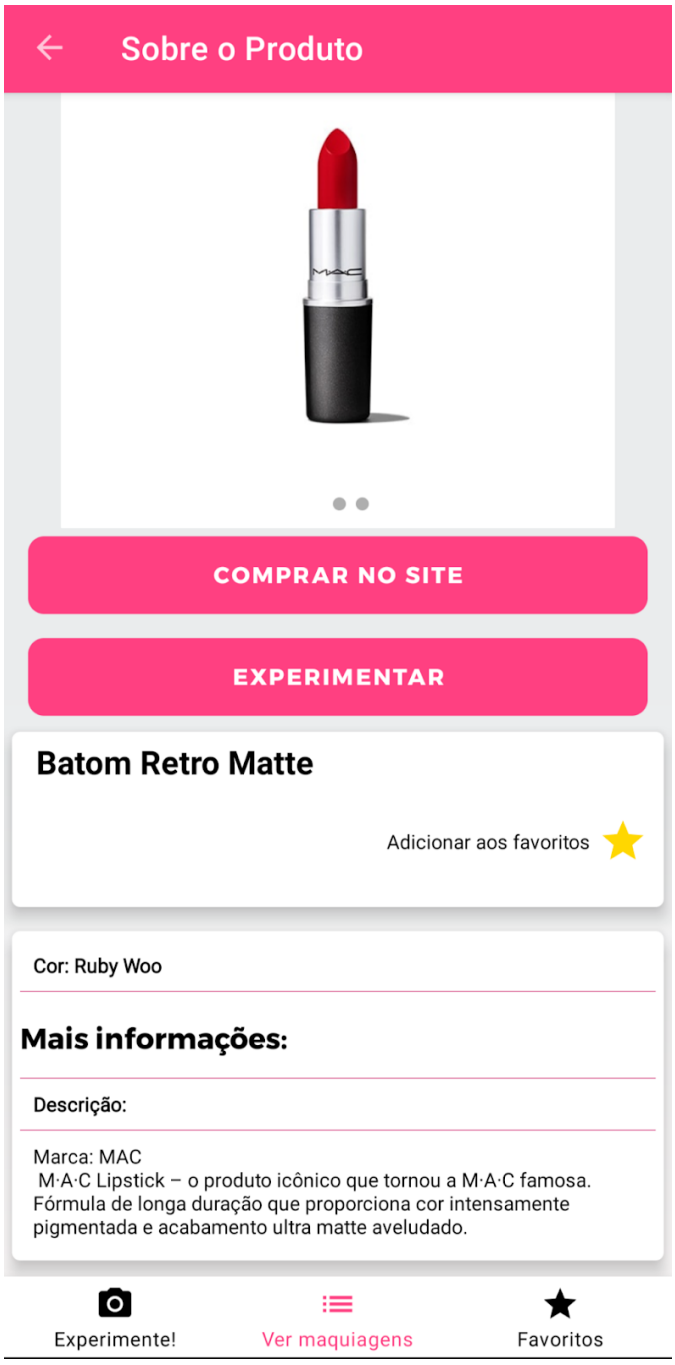
Fonte: Autor

Foi criada uma classe *ProductLoader* que é responsável por carregar os produtos e é chamada pelas *activities*. Como dito anteriormente, este trabalho não possui servidor nem banco de dados, então os produtos são *hard-coded*. Como a responsabilidade de carregar os produtos foi isolada nesta classe, caso a forma de carregar os produtos mude - passe a vir de um servidor por exemplo - precisaria mudar apenas o método desta classe, sem precisar mudar as *activities*.

5.2.2 PDP do MaquiAR

A figura abaixo representa a página de descrição do produto do MaquiAR:

Figura 28 - Tela descrição do produto do MaquiAR



Fonte: Autor

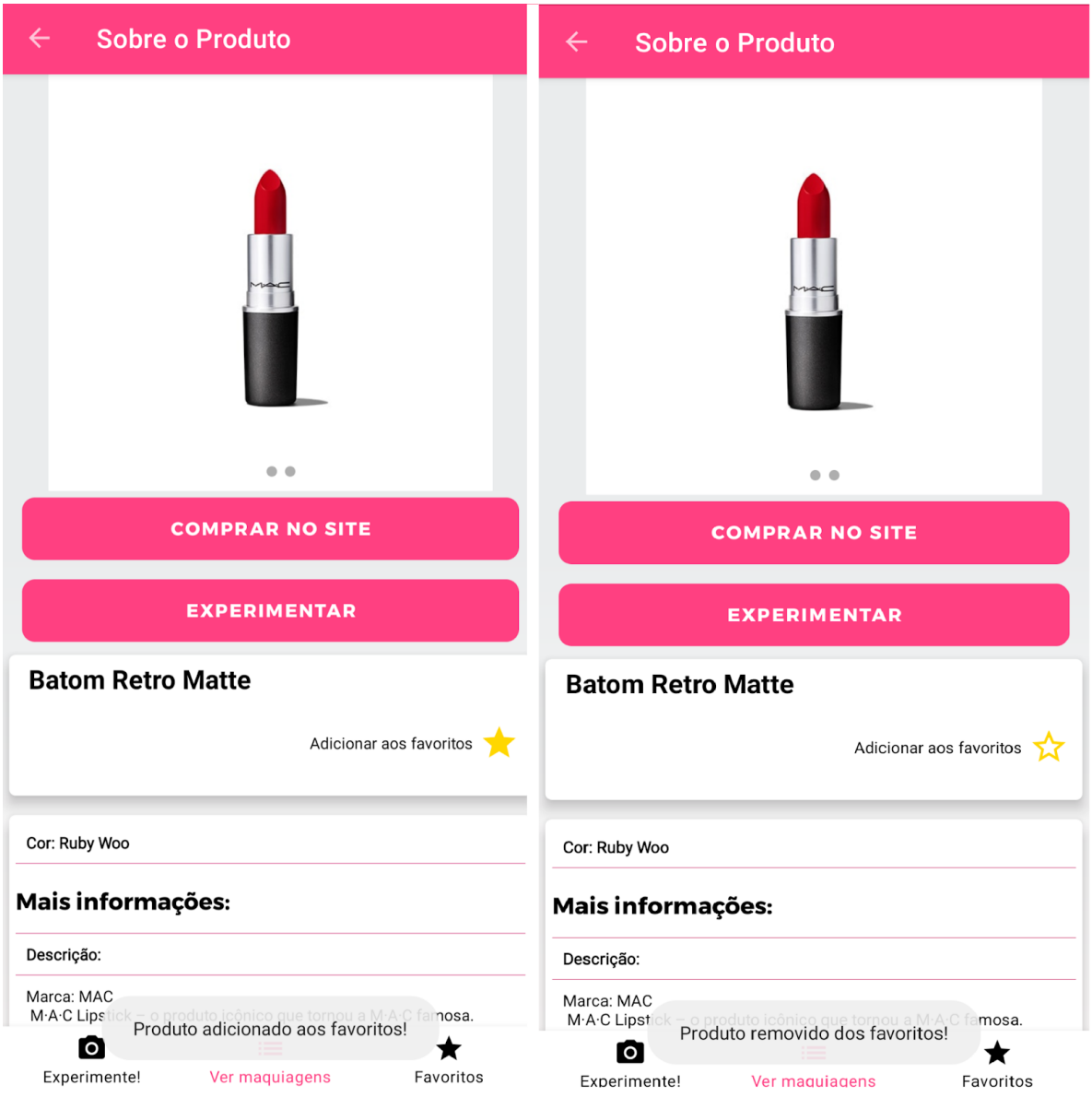
Para a exibição das fotos do produtos foi utilizada a biblioteca *Android Image Slider*[4]. O botão “comprar no site” direciona o usuário para o site onde o produto pode ser comprado (requisito funcional essencial 7).

[4] https://github.com/denzcoskun/ImageSlideshow - Acesso em junho de 2022

5.2.3 Favoritos

A funcionalidade de adicionar e remover produtos dos favoritos é acionada pelo usuário ao clicar no ícone de estrela na tela PDP e na tela de *try-on*. O ícone de estrela possui dois estados, vazado e preenchido. Quando o ícone está vazado, significa que o produto não pertence a lista de favoritos do usuário e quando o ícone está preenchido significa que pertence. Ao clicar no ícone o estado é alternado e uma mensagem é enviada ao usuário através de um *Toast.*

Figura 29 - Usuário alternando o botão de favorito



Fonte: Autor

Enquanto as *activities* da PDP e da tela de *try-on* são quem captam o desejo do usuário de adicionar ou remover um produto dos favoritos, é a classe *FavoriteUtils* que executa de fato esse processo. Ela é uma classe cuja responsabilidade é cuidar das lógicas relacionadas aos favoritos: carregar os produtos favoritos, salvar os favoritos, adicionar um produto aos favoritos, remover um produto dos favoritos e checar se um produto está nos favoritos.

A estratégia utilizada para salvar os favoritos foi armazenar os ids dos produtos no *App preferences.* O *preferences* permite armazenar dados do tipo chave-valor exclusivos de um aplicativo, esses dados não são acessíveis por outros aplicativos e são excluídos quando o aplicativo é desinstalado. (GOOGLE, 2022c). Para fazer com que a lista de ids se torne armazenável no *preferences*, ela é transformada em um JSON e salva com a chave “Favoritos”. Ao recuperar o valor, esse JSON é transformado novamente na lista de ids.

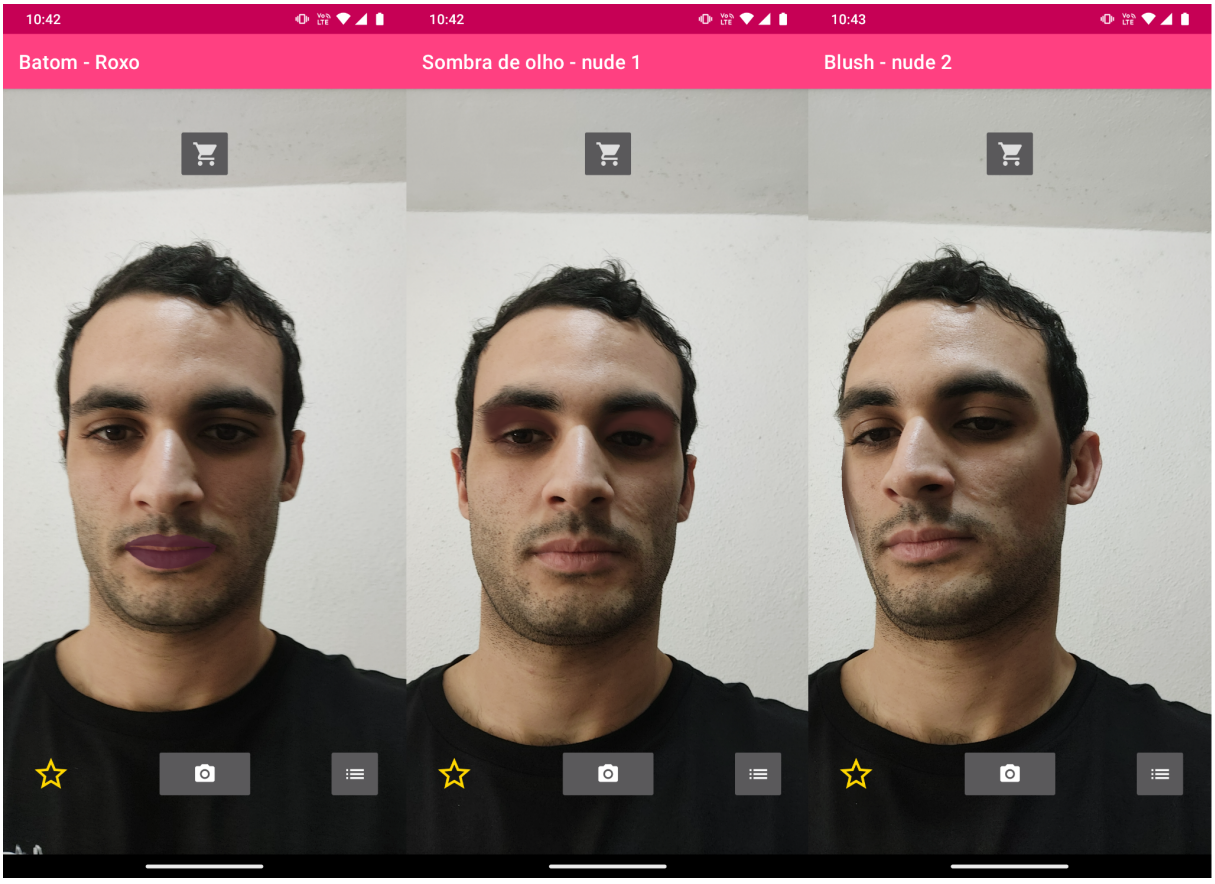
5.2.4 Visualizar os produtos com Realidade Aumentada

Como citado anteriormente, foi utilizada a biblioteca *ARCore Augmented Faces wrapper without Sceneform.* Foi citado como utilizar a biblioteca para adicionar textura e cores, mas não como ela faz para adicioná-las à imagem da câmera. Aqui entra o conceito de *shaders*. *Shaders* são um conjunto de instruções que são executadas todas ao mesmo tempo para cada pixel da tela. O programa funciona como uma função que recebe uma posição e retorna a respectiva cor. (VIVO; LOWE, 2015). A textura é adicionada e renderizada pela classe *AugmentedFaceRenderer.*

Como a biblioteca está escrita, porém, teríamos que criar uma textura diferente para cada cor de produto. Por isso, foi utilizada uma estratégia para que pudéssemos reaproveitar a mesma textura para todos os produtos do mesmo tipo. O primeiro passo foi criar a textura com a cor branca, pois ela na OpenGL Shading Language (GLSL) possui os valores (1,1,1), sendo o elemento neutro da multiplicação. No arquivo *object.frag*, que se refere às propriedades de um *fragment shader*, definimos a cor do *object* como a cor da textura multiplicada pela cor que

passamos como parâmetro, assim o resultado final será a cor que passamos como parâmetro.

Figura 30 - Filtros de batom, sombra de olho e blush do MaquiAR



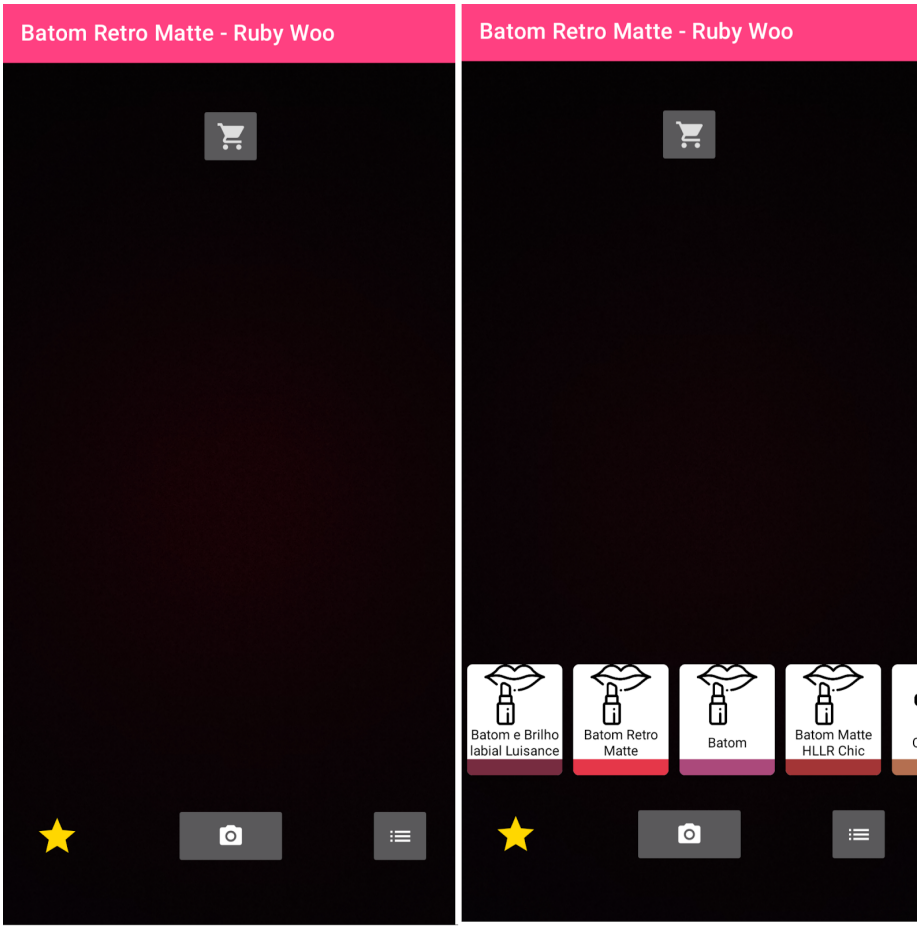
Fonte: Autor

5.2.5 Mudar de produto que está sendo experimentado sem sair da tela de RA

Esta funcionalidade é de suma importância para a experiência do usuário. Para implementá-la, precisamos de executar duas etapas: exibir a lista de produtos, para que o usuário possa escolher, e mudar de produto quando um for escolhido.

Para a primeira etapa foi decidido que a lista começa oculta, e aparece quando o usuário pressionar o botão com o ícone “lista”. Desta forma ela não ocupa grande parte da tela quando não é de desejo do usuário visualizá-la. Para a lista, foi novamente usada uma *RecyclerView*, desta vez horizontal.

Figura 31 - Tela try-on com e sem lista de produtos do MaquiAR



Fonte: Autor

Para a segunda etapa foi adicionado uma *flag* que indica se outro produto foi escolhido pelo usuário. No método *onFaceUpdate* a *flag* é checada e caso outro produto tenha sido escolhido atualiza-se tanto a textura quanto a cor e muda a *flag* para *false*. Para que a classe reconheça qual produto foi escolhido, foi criada a interface *OnProductSelectedListener,* feito com que a *ARVisualizationAcitivty* a implementasse e usada no *setOnClickListener* do *recycler*. Na implementação da *activity,* o produto é atualizado e a *flag* muda para *true.*

5.2.6 Tirar e compartilhar fotos na visualização com RA

Esta funcionalidade foi a que mais custou tempo para ser implementada. Isso se deu pois os métodos mais comuns de capturar a tela do aplicativo não funcionam com a *GLSurfaceView*, que a classe *AugmentedFaceFragment* utiliza. Isso se dá pois a *SurfaceView* possui duas partes, a *Surface* e a *View*. A *Surface* fica

em uma camada diferente dos outros elementos *View*. Assim, métodos como *canvas*, ou *getDrawingCache* não funcionam, pois eles capturam apenas a camada *View*. Assim o resultado era uma tela preta.

A solução encontrada foi adicionar uma *flag* na classe *AugmentedFaceFragment* para indicar se a tela da *Surface* deve ser capturada. Essa *flag* é checada no fim do método *onDrawFrame,* ou seja, depois de todos os elementos terem sido adicionados, e caso seja *true* o bitmap da *Surface* é capturado, salvo e a *Intent* de compartilhar imagem é chamada.

6. EXPERIMENTOS E RESULTADOS

Foi realizado um experimento com o objetivo de avaliar a recepção do aplicativo pelos usuários, se o uso dele aumentaria a intenção de compra de uma maquiagem, e testar a usabilidade do mesmo. O experimento será detalhado na seção seguinte.

6.1 Experimento: Teste com usuários

Foram convidados 10 estudantes da Universidade Federal de Santa Catarina, para participar da pesquisa. O experimento foi composto por quatro etapas: um questionário pré-uso do aplicativo, a utilização do aplicativo pelo participante, um questionário pós-uso do aplicativo e o questionário *System Usability Scale* (SUS).

O experimento foi realizado em Junho de 2022 e teve duração total de aproximadamente 2h, sendo a média de duração do teste de cada participante aproximadamente 10 minutos. O participante abria, através de um código QR uma página do *Google Forms* que possuía todos os questionários, assim como algumas instruções. Após responder o questionário pré-uso o participante recebia o dispositivo (Motorola One Hyper com Android versão 11) com o MaquiAR instalado para utilizar o aplicativo. Após o uso do aplicativo, o participante respondia os questionários restantes.

O formulário pré-uso possui perguntas com o objetivo de avaliar a familiaridade do participante com compras de maquiagem online. Foram elaboradas perguntas diretas, com o objetivo de validar o problema.

A etapa de utilização do aplicativo pelo participante acontece sem interferência do condutor da pesquisa, ou seja, o participante não teve “dicas” de como utilizar o aplicativo. A única instrução que fora passada foi de que o participante deveria experimentar ao menos duas maquiagens diferentes. Essa decisão foi feita para que a usabilidade do aplicativo também seja testada.

O formulário pós-uso possui perguntas para avaliar a recepção do MaquiAR pelo usuário. Foram elaboradas 4 perguntas, utilizando a escala Likert de 5 pontos, e após cada uma havia um espaço livre para que o participante escrevesse o que quisesse relacionado a pergunta.

Por fim, com o objetivo de avaliar a usabilidade do MaquiAR, foi aplicado o SUS. O SUS é um questionário feito para medir a percepção de usabilidade. Ele foi criado por John Brooke em 1986 e é o questionário padrão para a indústria, sendo referenciado por mais de 600 publicações. (SAURO, 2016). Ele é constituído por 10 afirmações, cada uma com uma escala Likert de 5 pontos de concordância. É importante que a ordem original das perguntas seja mantida, para não afetar no resultado final.

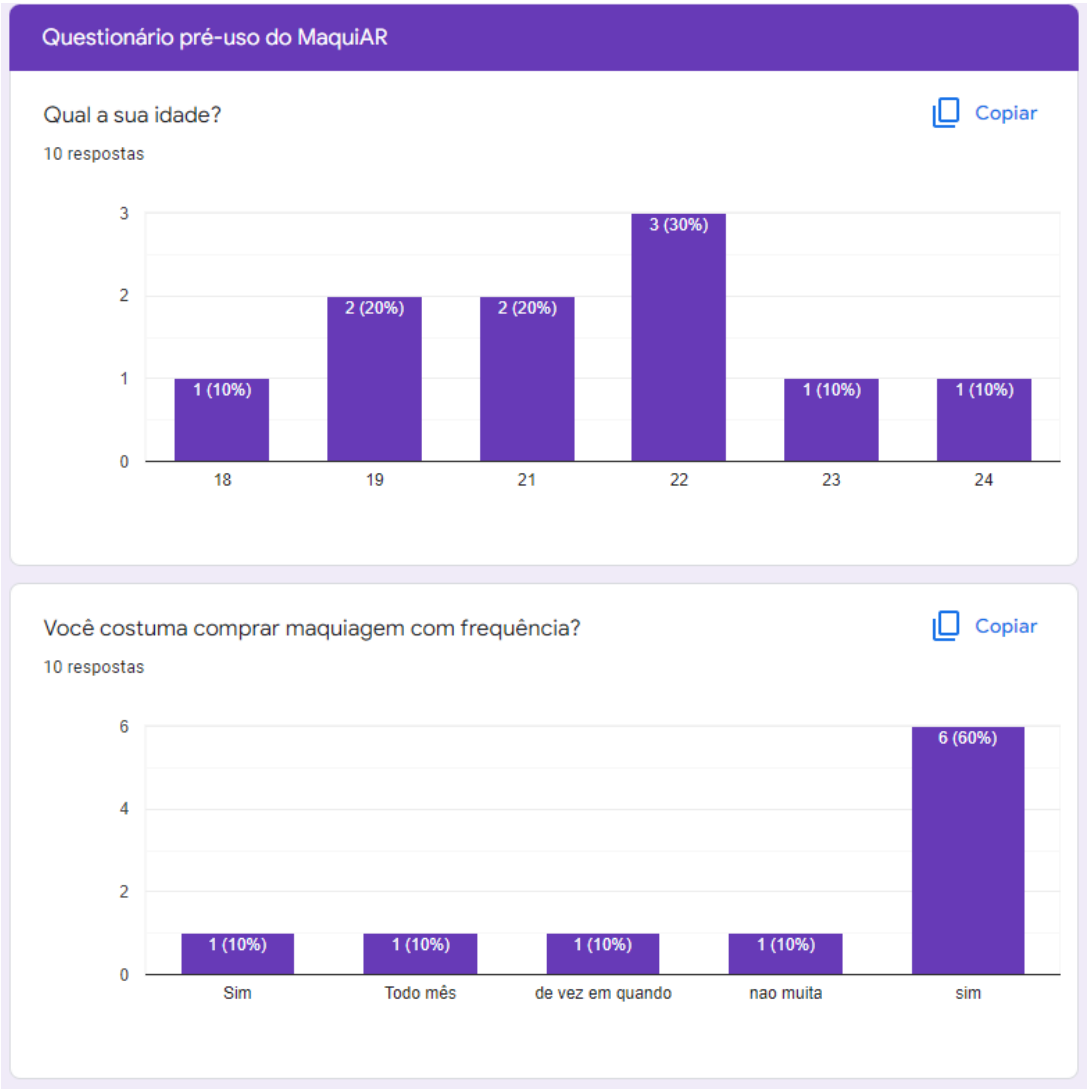
Para calcular o resultado do SUS, é necessário para cada item ímpar subtrair 1 a resposta do usuário, enquanto os pares o score é 5 menos a resposta do usuário. As afirmações ímpares são de teor positivo em relação à aplicação. Neste caso, é desejável que as respostas dos usuários concordem com as afirmações. Por outro lado, as afirmações pares são negativas e, portanto, para serem bem avaliadas o ideal é que os usuários respondam tendendo a discordar com a afirmação. Obtendo o valor para cada item, soma-se tudo e multiplica por 2.5. Esse valor final vai resultar em um valor entre 0 a 100. Sauro (2016) analisou dados de mais de 5000 usuários que avaliaram 500 estudos diferentes e a média de pontuação foi 68. Ele afirma que uma pontuação acima de 68 deve ser considerada acima da média e qualquer pontuação menor abaixo da média.

Os questionários do experimento estão no apêndice A.

6.2 Resultados

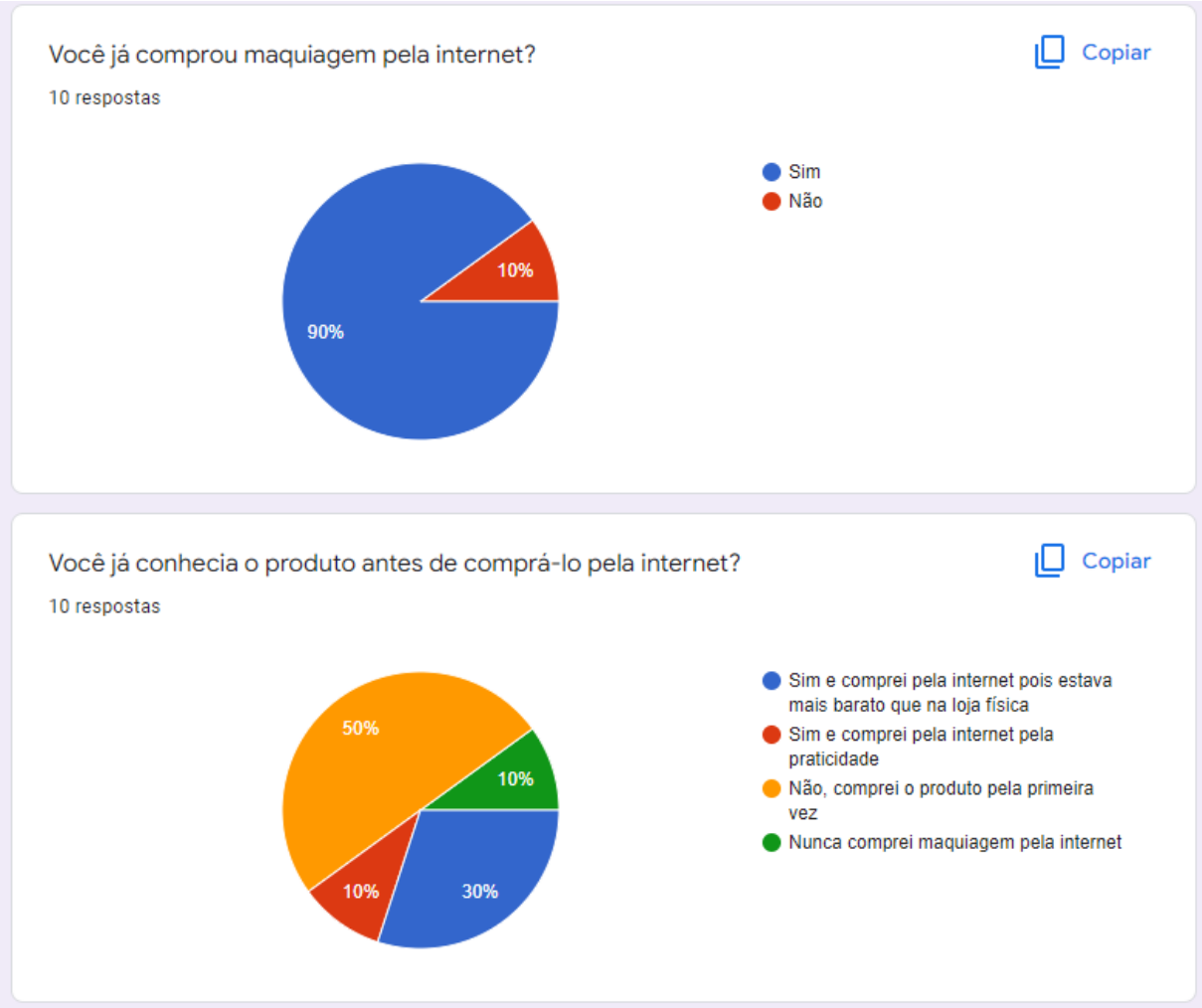
Todos os questionários foram aplicados utilizando *Google Forms*, e com isto, os resultados gráficos foram produzidos automaticamente. Os resultados do questionário pré-uso do MaquiAR será apresentado a seguir:

Figura 32 - Resultados do questionário pré-uso parte 1



Fonte: Autor

Figura 33 - Resultados do questionário pré-uso parte 2

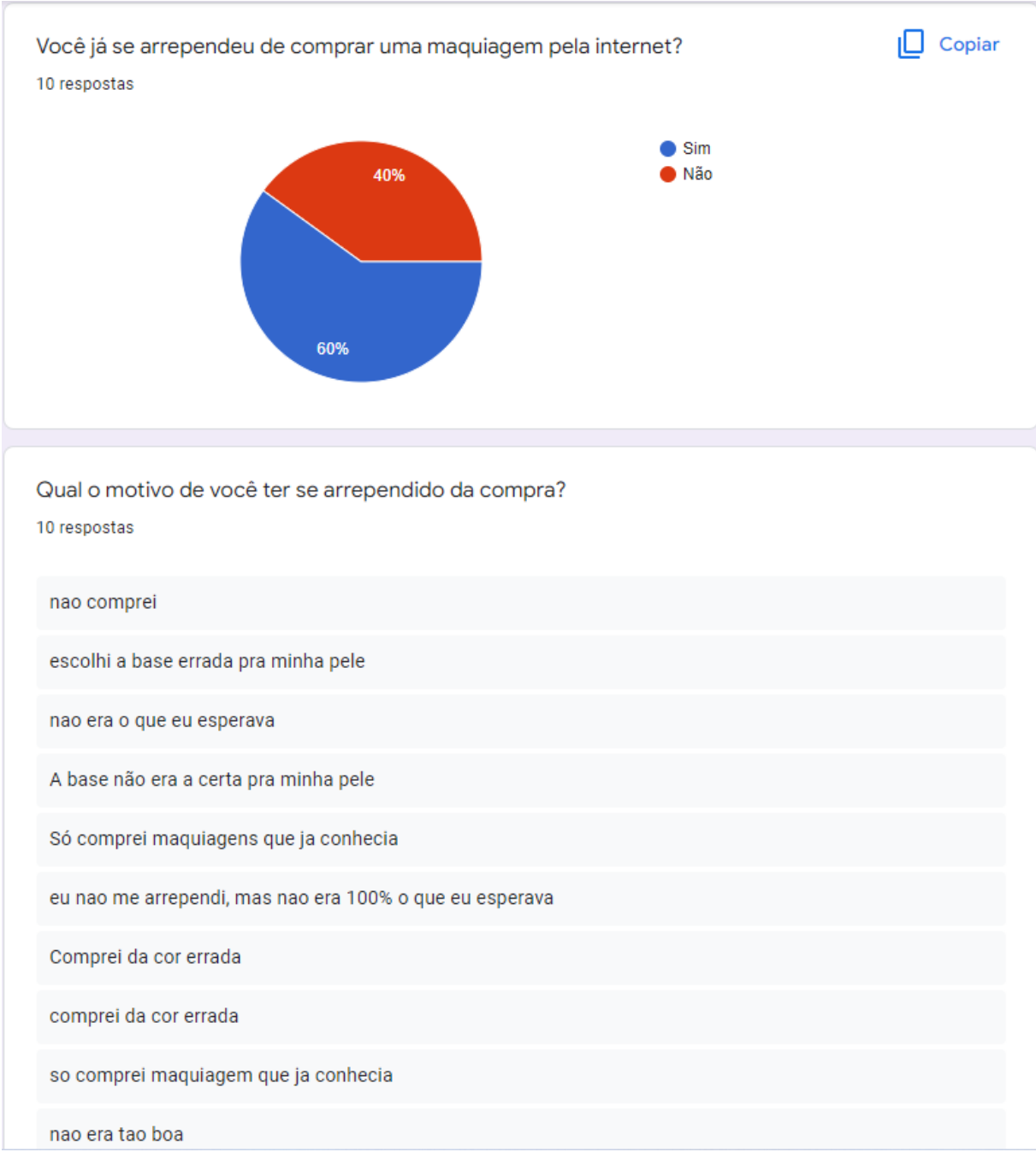


Fonte: Autor

Podemos observar que os participantes da pesquisa são jovens (entre 18 e 23 anos) e costumam comprar maquiagem com uma certa frequência. A grande maioria já comprou maquiagem pela internet, apenas um respondeu não.

Em relação à pergunta “Você já conhecia o produto antes de comprá-lo pela internet?”, houve uma falha na formulação da pergunta por parte do autor. O objetivo da pergunta era saber se o participante já comprou algum produto que não conhecia antes, porém, da maneira que foi formulada a pergunta deixou aberta para diferentes interpretações. Isso se dá pois os participantes podem ter comprado maquiagem online múltiplas vezes, algumas com produtos já conhecidos, outras de novos produtos.

Figura 34 - Resultados do questionário pré-uso parte 3

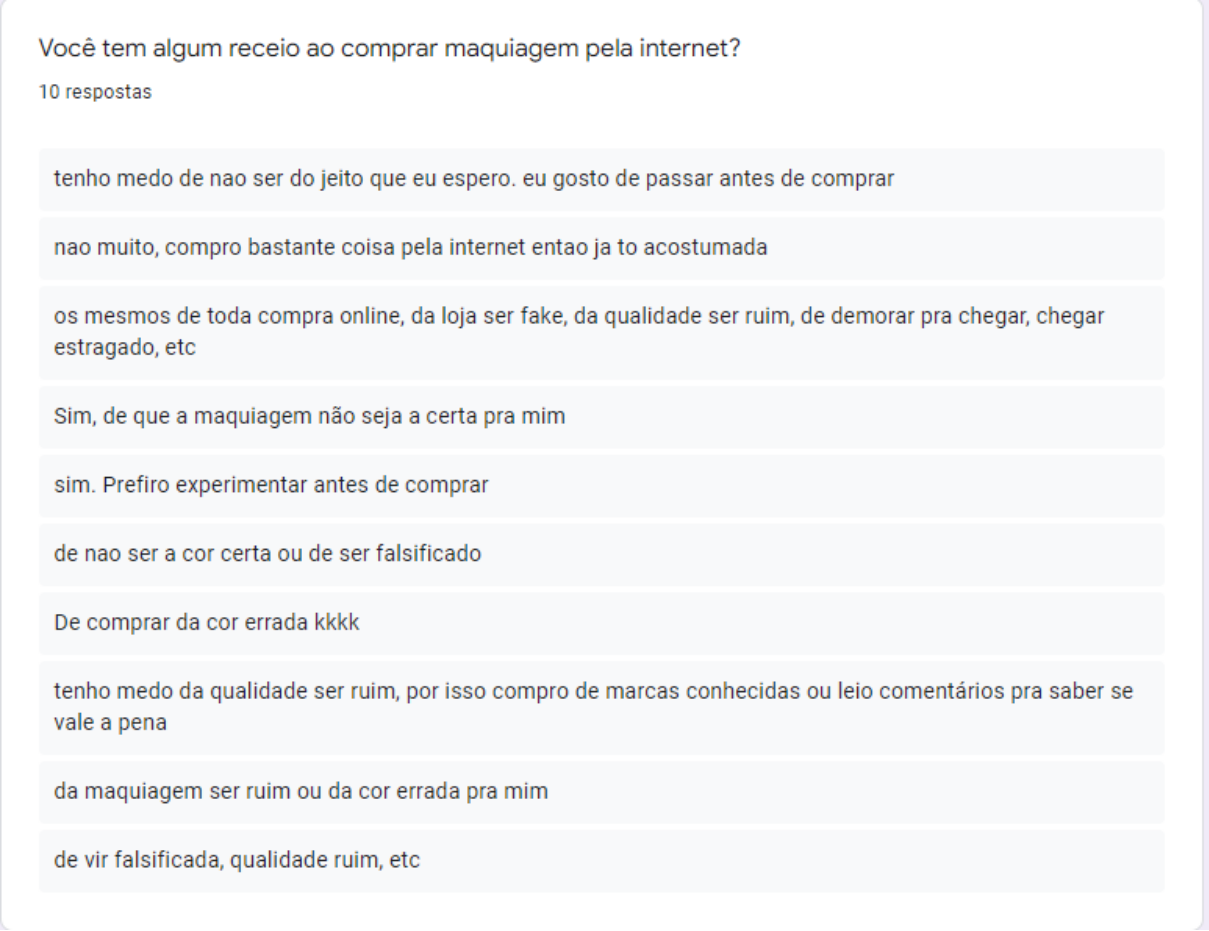


Fonte: Autor

A maioria dos participantes afirmaram ter se arrependido de uma compra online de maquiagem. Ao analisarmos os que afirmaram não ter se arrependido, dois deles se dão pelo fato do participante nunca ter comprado maquiagem online, outro pois sempre comprou produtos que já conhecia e o último afirmou que apesar de não ter se decepcionado, o produto não era exatamente o que esperava. Dessa forma, podemos dizer que os participantes que compraram uma maquiagem pela primeira vez de forma online acabaram se arrependendo. Dentre os motivos do

arrependimento da compra destaca-se a cor do produto não ser a ideal para a pessoa.

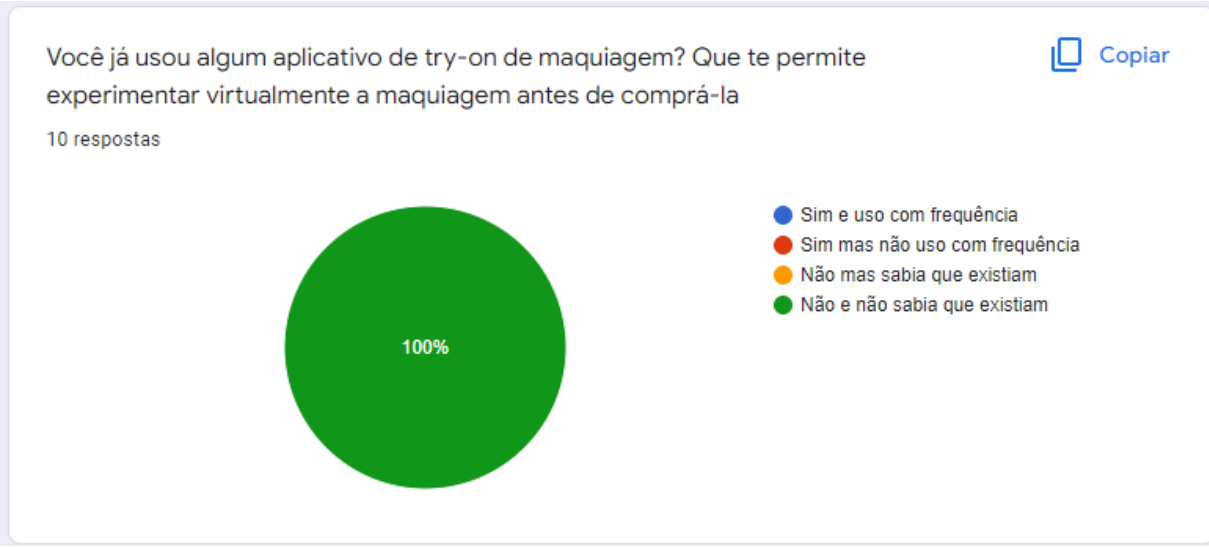
Figura 35 - Resultados do questionário pré-uso parte 4



Fonte: Autor

Todos os participantes afirmaram possuir algum receio de comprar maquiagem pela internet. Foram citados receios que são comuns de qualquer compra online, como da loja ou do produto ser falsos ou de chegar estragado. Foram também listados receios específicos do contexto de maquiagens: a cor e a qualidade do produto.

Figura 36 - Resultados do questionário pré-uso parte 5



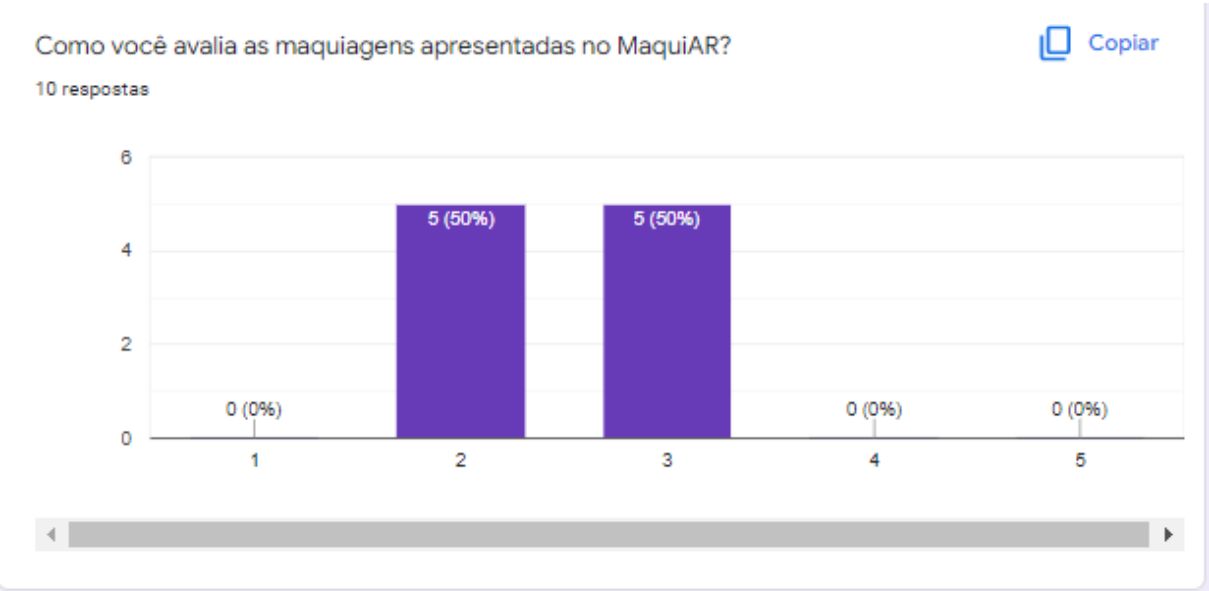
Fonte: Autor

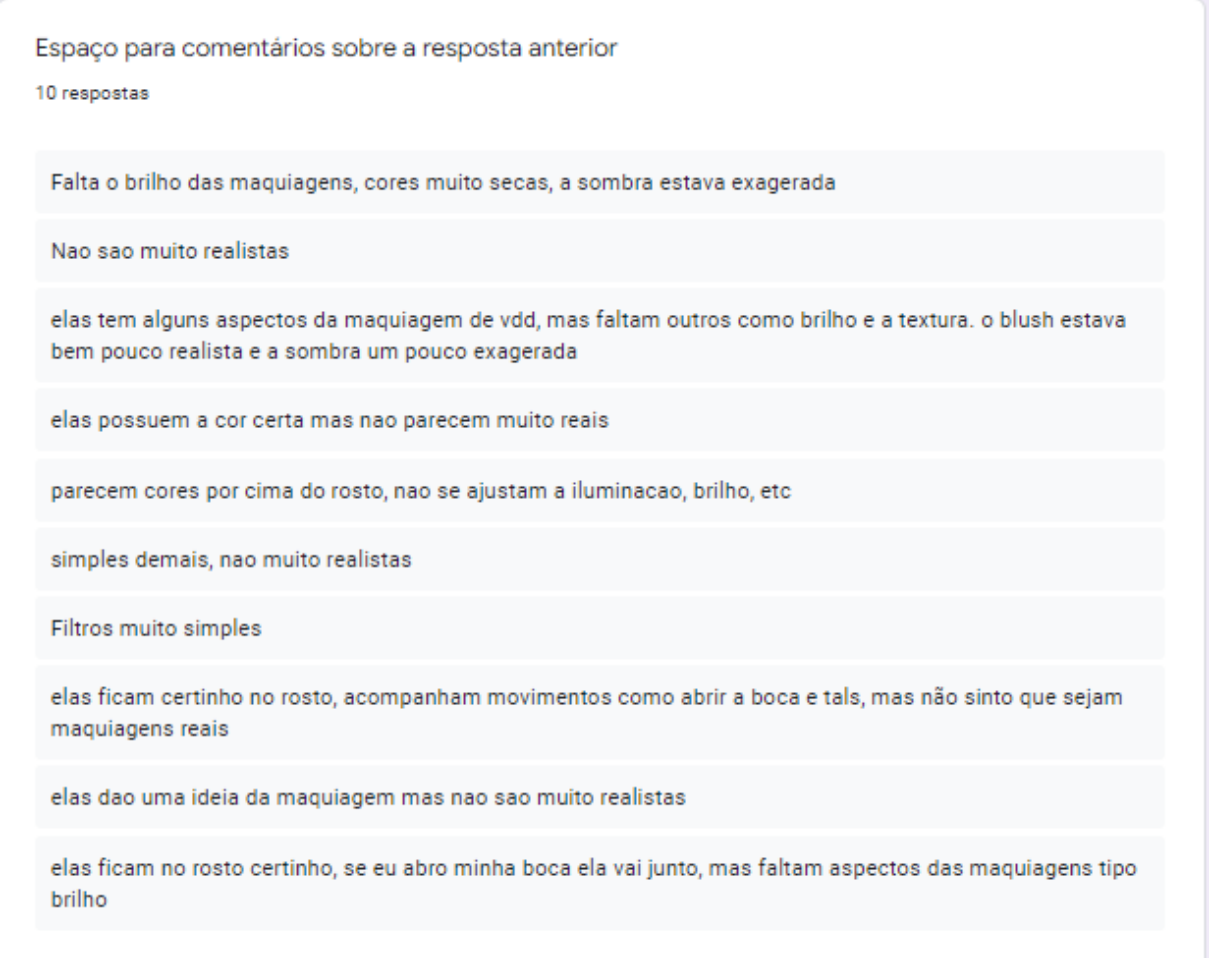
Nenhum participante afirmou conhecer aplicativos de *try-on* de maquiagem. Num primeiro olhar, esse resultado é surpreendente, mas quando lembramos dos aplicativos do mercado - analisados na seção 3 deste trabalho - percebemos que os principais aplicativos de comércio eletrônico de maquiagens não possuem *try-on*, e que os aplicativos de *try-on* de maquiagem não são muito utilizados para esse fim no Brasil, seja por não possuir produtos e lojas brasileiras, ou por ser mais usado como um aplicativo de edição de fotos. Um fato que podemos destacar é o desconhecimento, por parte dos participantes, de que o Instagram possui essa funcionalidade.

Ao analisar as respostas da primeira etapa podemos afirmar que o problema foi validado.

As figuras seguintes são referentes aos resultados do questionário pós-uso do MaquiAR:

Figura 37 - Resultados do questionário pós-uso parte 1



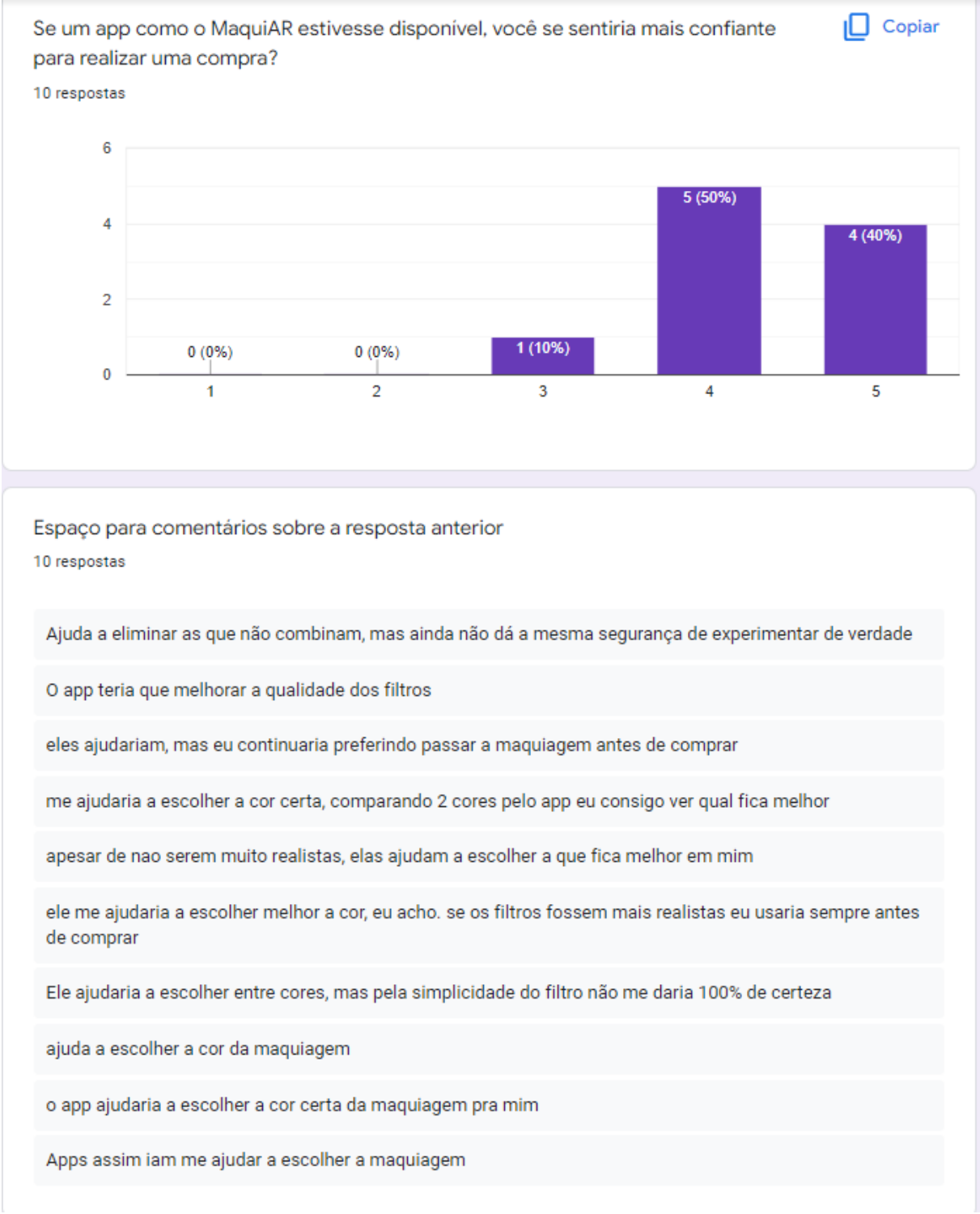


Fonte: Autor

Em geral, as maquiagens apresentadas no MaquiAR foram avaliadas como pouco realistas. Em relação aos motivos da avaliação, dois participantes disseram

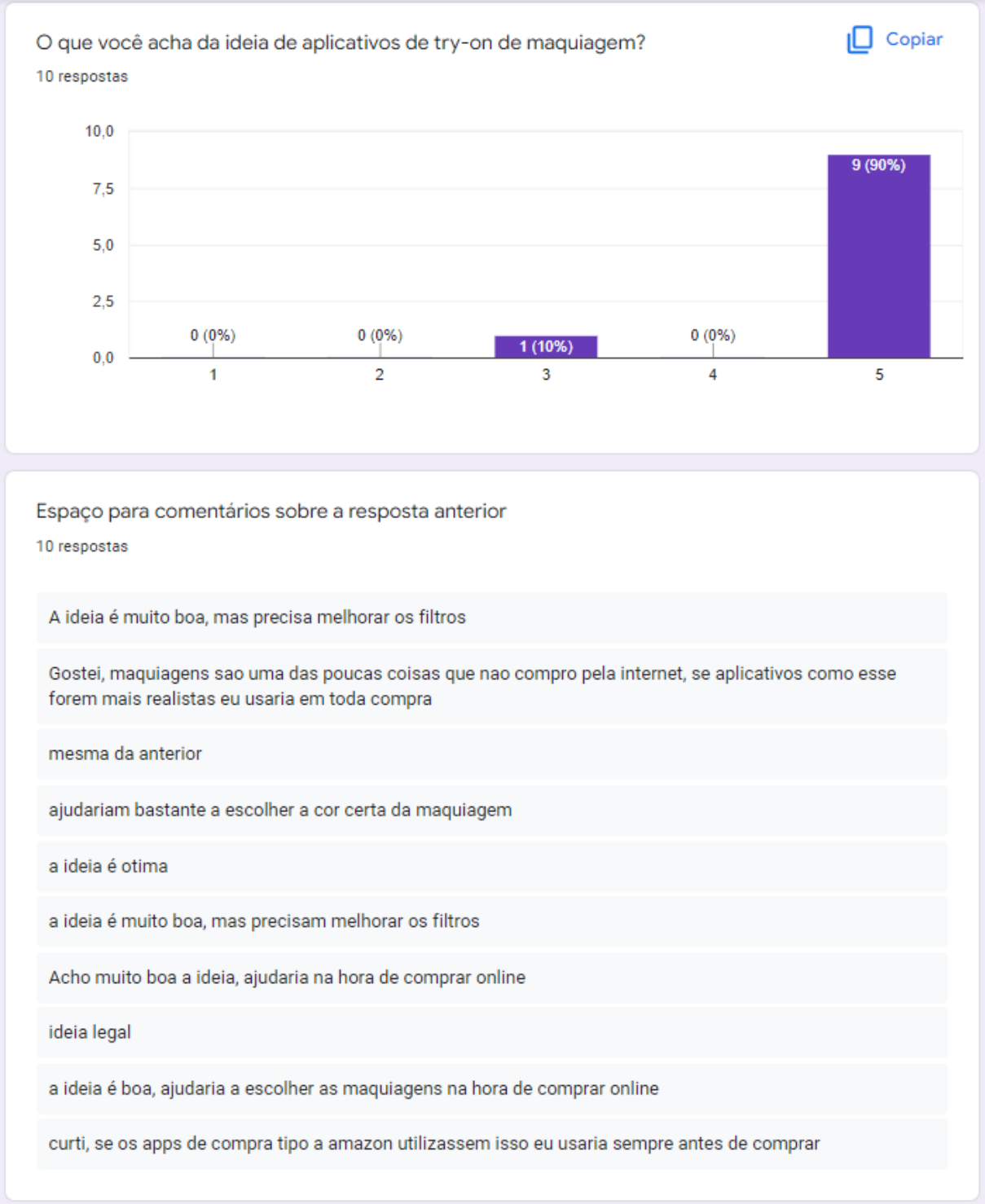
que elas eram “muito simples”, enquanto outros detalharam melhor, citando a falta de ajuste ao brilho e à iluminação.

Figura 38 - Resultados do questionário pós-uso parte 2



Fonte: Autor

Figura 39 - Resultados do questionário pós-uso parte 3

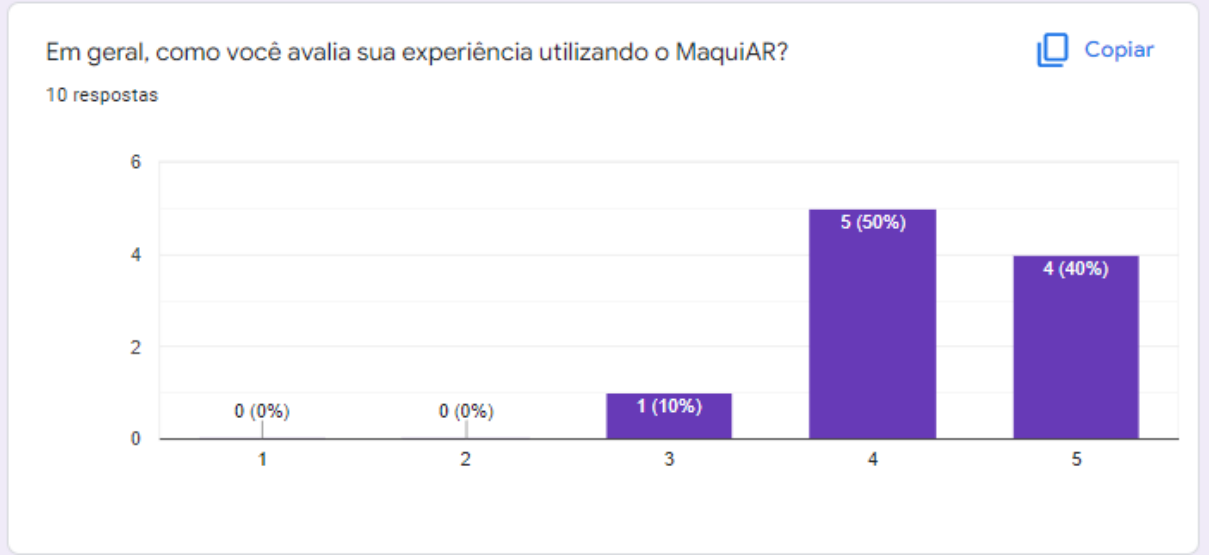


Fonte: Autor

A ideia de um aplicativo como o MaquiAR foi vista positivamente pelos participantes. A grande maioria dos participantes afirmaram que caso um aplicativo como o MaquiAR estivesse disponível eles se sentiriam mais confiantes para realizar uma compra, tendo afirmações de que usariam toda vez que fossem comprar

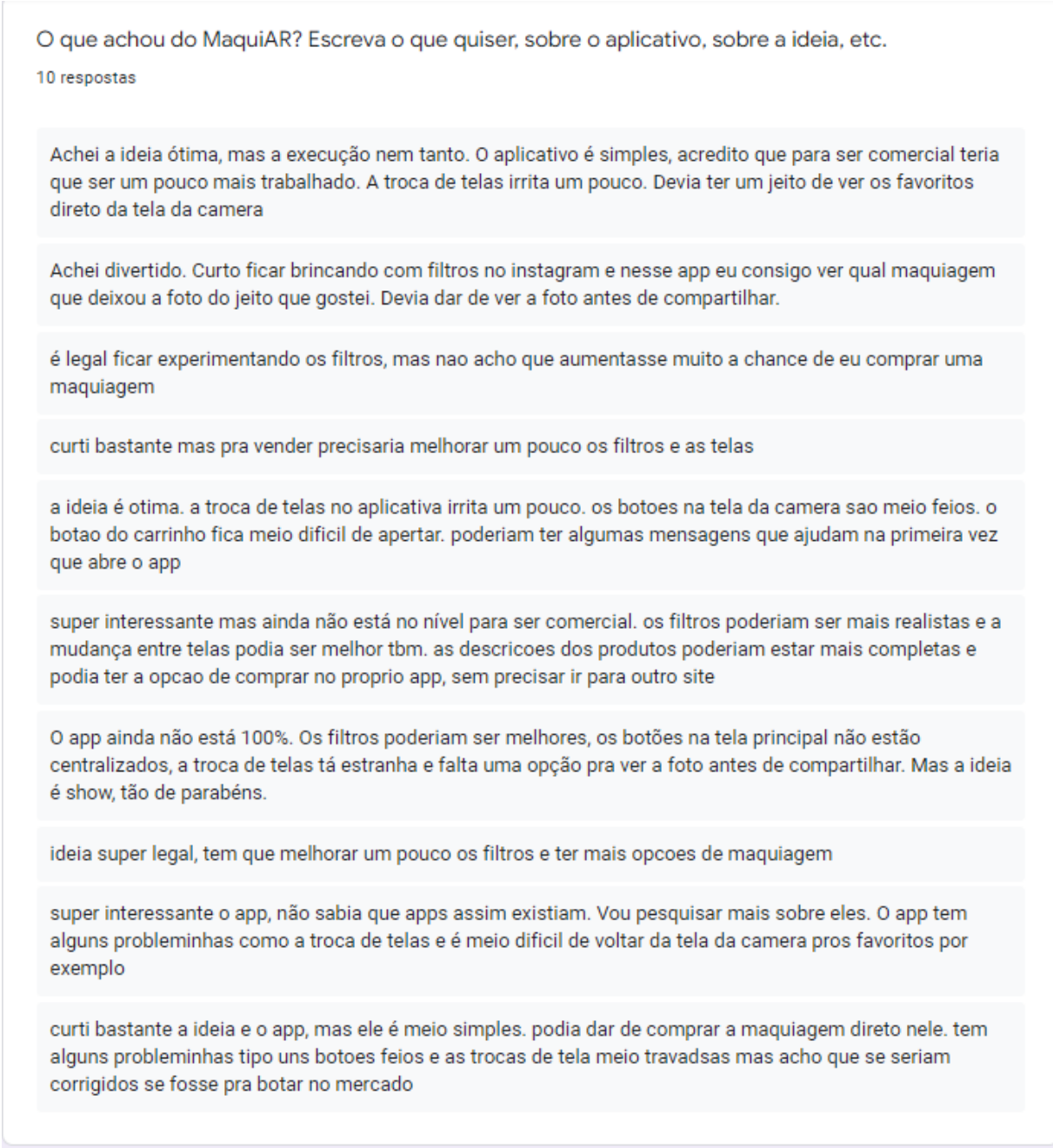
maquiagem online. Foi comentado também que, apesar dos filtros de maquiagem não serem muito realistas, é possível identificar qual cor seria a melhor para o usuário comprar. Como vimos anteriormente, “cor errada” foi o motivo de arrependimento mais citado pelos participantes.

Figura 40- Resultados do questionário pós-uso parte 4



Fonte: Autor

Figura 41 - Resultados do questionário pós-uso parte 5



Fonte: Autor

Sobre a experiência utilizando o aplicativo, os participantes afirmaram que ela foi positiva. Foi comentado por múltiplos usuários que a ideia é ótima mas que para ser comercializada precisa melhorar alguns pontos como o realismo dos filtros e a usabilidade. Dentre as queixas sobre a usabilidade do aplicativo estão a falta de fluidez na troca de telas - problema causado por que as telas foram implementadas como *activities* e não *fragments* - e o botão de descrição do produto na tela de *try-on*.

A tabela abaixo representa o resultado do questionário SUS:

Tabela 3 - Resultado questionário SUS

Afirmação	Discordo totalmente	Discordo parcialmente	Nem concordo, nem discordo	Concordo parcialmente	Concordo totalmente
Eu penso que usarei este sistema com frequência	0	1	0	2	7
Acho o sistema desnecessariamente complexo	7	2	1	0	0
Penso que o sistema é fácil de usar	0	0	1	3	6
Acho que vou precisar de ajuda de um técnico para usar este sistema	10	0	0	0	0
Acho as funções deste sistema bem integradas	0	0	2	8	0
Encontro muitas inconsistências neste sistema	2	7	1	0	0
Imagino que as pessoas aprenderão rapidamente a usar este sistema	0	0	0	0	10
Acho o sistema imprático de usar	3	7	0	0	0
Senti-me confiante ao usar o sistema	0	0	1	5	4
Precisei aprender muitas coisas antes de ser capaz de operar o sistema	9	1	0	0	0

Fonte: Autor

É possível observar que os usuários não tiveram grandes dificuldades na utilização do aplicativo. Retomando o que fora explicado na seção 6.1, para calcular o resultado do questionário é preciso:

- Para cada item ímpar subtrair 1 da resposta do usuário
- Para cada item par o resultado é 5 menos a resposta do usuário
- Após feito os passos acima, soma-se tudo e multiplica por 2,5.

Aplicando as três etapas, o resultado obtido foi uma média de 88,5 entre os 10 usuários. Isso posiciona a usabilidade do MaquiAR acima da média da usabilidade dos aplicativos.

7. CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve o propósito de desenvolver um aplicativo móvel com realidade aumentada voltada para o *e-commerce* de maquiagens. Inicialmente, foi estudado, através de uma revisão de literatura, os impactos do uso de realidade aumentada na apresentação de produtos na decisão de compra de um usuário. Foram, então, analisados os aplicativos no mercado relacionados a *e-commerce* e *try-on* de maquiagem. Com essa análise foi possível perceber qualidades e defeitos de outros aplicativos para levantar os requisitos do sistema deste trabalho. Os requisitos funcionais e não funcionais delinearam a construção do sistema. Depois de concluído o desenvolvimento aplicativo, o mesmo foi testado por usuários reais por meio de um experimento.

Como objetivos específicos, este trabalho possuía a análise das vantagens da utilização da RA na apresentação de um produto ao usuário. Este objetivo foi atingido na revisão da literatura. Outro objetivo específico era examinar a viabilidade da utilização de RA em diferentes tipos de produtos. Foi percebido que é possível utilizar a RA para diversos tipos de maquiagens, com algumas limitações, como a fidelidade limitada, e isso será discutido mais à frente nesta seção. Outro objetivo específico era conhecer o estado da arte em RA e suas aplicações no *e-commerce*. A primeira parte deste objetivo foi atingida na seção 2.4 e durante o desenvolvimento, já que foi usado uma ferramenta que faz parte do estado da arte, o ARCore. A segunda parte foi concluída ao analisarmos os aplicativos semelhantes na seção 3. Por último, este trabalho tinha como objetivo específico aprofundar o conhecimento no desenvolvimento de aplicativos móveis. Este objetivo foi alcançado, tendo este trabalho desenvolvido um aplicativo Android utilizando de diversos conceitos e ferramentas comumente utilizados pela indústria e seguindo padrões de desenvolvimento recomendados pelo guia do desenvolvedor Android.

Ao analisarmos os resultados obtidos pelo experimento, podemos tirar algumas conclusões. A recepção do uso da realidade aumentada na exibição de maquiagens foi muito positiva, tendo os participantes do experimento respondido que usariam aplicativos assim toda vez que fariam uma compra de maquiagem online. Outro ponto de destaque é que isso se deu mesmo com a avaliação de que as maquiagens apresentadas no MaquiAR não são muito realistas. Os motivos para tal avaliação não foram profundamente estudados, porém é importante salientar que os participantes utilizaram um dispositivo que não é o celular pessoal deles então a qualidade da câmera, diferente da que estão acostumados, pode ter afetado nas suas avaliações. Usuários afirmaram que apesar delas não terem algumas características da maquiagem real, as cores elas possuíam um grau de fidelidade que lhes trazia segurança para escolher a cor certa de um determinado produto.

Dessa forma, é esperado que, nos próximos anos, a realidade aumentada esteja cada vez mais presente com esse propósito, de apresentar produtos do *e-commerce* ao usuário, especialmente pois a tecnologia está se desenvolvendo e se tornando cada vez mais realista, o que trará ainda mais confiança ao comprador.

Obviamente, sempre há pontos de melhoria, em qualquer trabalho. Principalmente devido a restrição de tempo, algumas funcionalidades não foram implementadas. Segue sugestões de melhorias e de trabalhos futuros:

Sugestões de melhorias:

- Transformar as *activities* de listar produtos, PDP e favoritos em *fragments*, o que tornará a navegação no aplicativo mais fluida e melhorará a experiência do usuário.
- Tornar os botões da tela de *try-on* visualmente mais agradáveis.
- Filtrar os produtos, por categoria, cor, etc.
- Indicativo de qual produto está sendo experimentado na lista de produtos na tela de *try-on*.

Sugestões para trabalhos futuros:

- Implementar os requisitos funcionais desejáveis.
- Implementar servidor backend.
- Realizar o experimento para um grupo maior e mais heterogêneo

- Estudar os motivos dos participantes terem avaliado os filtros como não muito realistas
- Implementar sistema que utilize inteligência artificial para recomendar maquiagens ao usuário.

8. REFERÊNCIAS

APPLE. **ARKit Overview**. Disponível em: https://developer.apple.com/augmented-reality/arkit/. Acesso em: 22 fev. 2022.

ANTHONY. **Why It's Important to Sketch Before You Wireframe**. 2012. Disponível em: https://uxmovement.com/wireframes/why-its-important-to-sketch-before-you-wireframe/#:~:text=Sketching%20Makes%20You%20Think%20and,can%20draw%20them%20on%20paper.. Acesso em: 16 jun. 2022.

BARRIENTOS, Jess. **RecyclerView or ListView? : Pros, cons, and examples with Kotlin**. 2020. Disponível em: https://dev.to/jbc7ag/recyclerview-or-listview-pros-cons-and-examples-with-kotlin-2nb2. Acesso em: 17 jun. 2022.

BECK, Marie; CRIÉ, Dominique. I virtually try it … I want it ! Virtual Fitting Room: a tool to increase on-line and off-line exploratory behavior, patronage and purchase intentions. **Journal Of Retailing And Consumer Services**, [S.L.], v. 40, p. 279-286, jan. 2018. Elsevier BV. http://dx.doi.org/10.1016/j.jretconser.2016.08.006.

BLOOMENTHAL, Andrew. Electronic Commerce (Ecommerce). **Investopedia**. 2021. Disponível em: https://www.investopedia.com/terms/e/ecommerce.asp. Acesso em: 24 fev. 2022.

ENGLAND, Rachel. Amazon's 'Live Mode' lets you try on makeup via its app. **Engadget**. 2019. Disponível em: https://www.engadget.com/2019-06-04-amazons-live-mode-lets-you-try-on-makeup-via-its-app.html. Acesso em: 20 fev. 2022.

E-COMMERCE BRASIL. **E-commerce no Brasil bate recorde e atinge R$ 53 bilhões no 1º semestre, mostra Ebit|Nielsen**. 2021. Disponível em:

https://www.ecommercebrasil.com.br/noticias/e-commerce-no-brasil-bate-recorde-e-atinge-r-53-bilhoes-ebit-nielsen-webshoppers/. Acesso em: 24 fev. 2022.

GOOGLE. **Conceitos fundamentais**. 2022a. Disponível em: https://developers.google.com/ar/develop/fundamentals. Acesso em: 22 fev. 2022.

GOOGLE. **Data and file storage overview**. 2022b. Disponível em: https://developer.android.com/training/data-storage. Acesso em: 16 jun. 2022.

GOOGLE. **Dispositivos compatíveis com ARCore**. 2022c. Disponível em: https://developers.google.com/ar/devices. Acesso em: 22 fev. 2022.

GOOGLE. **Introdução de rostos aumentados.** 2022d. Disponível em: https://developers.google.com/ar/develop/augmented-faces. Acesso em: 11 de mar. 2022.

INSTAGRAM. **As compras estão a chegar ao Instagram**. 2016. Disponível em: https://business.instagram.com/blog/shopping-on-instagram/. Acesso em: 20 fev. 2022.

INSTAGRAM. **A Better Shopping Experience on Instagram**. 2017. Disponível em: https://business.instagram.com/blog/a-better-shopping-experience-on-instagram. Acesso em: 20 fev. 2022.

INSTAGRAM. **Introducing Shopping Stickers to Instagram Stories**. 2018. Disponível em: https://about.instagram.com/blog/announcements/introducing-shopping-on-instagram-stories. Acesso em: 20 fev. 2022.

INSTAGRAM. **Introducing: Shops on Instagram**. 2020. Disponível em: https://business.instagram.com/blog/introducing-shops-on-instagram?locale=pt_BR. Acesso em: 20 fev. 2022.

INSTAGRAM. **O que são as compras no Instagram?** 2022. Disponível em: https://business.instagram.com/shopping. Acesso em: 20 fev. 2022.

KIM, Jiyeon; FORSYTHE, Sandra. Adoption of Virtual Try-on technology for online apparel shopping. **Journal Of Interactive Marketing**, [S.L.], v. 22, n. 2, p. 45-59, jan. 2008. Elsevier BV. http://dx.doi.org/10.1002/dir.20113.

LAMP, Amy. **Why is sketching such an important aspect of design?** 2011. Disponível em: https://crowdfavorite.com/why-is-sketching-such-an-important-aspect-of-design/. Acesso em: 16 jun. 2022.

LEONNARD, Leonnard; PARAMITA, Annisa S.; MAULIDIANI, Jasmine J.. The Effect of Augmented Reality Shopping Applications on Purchase Intention. **Esensi**: Jurnal Bisnis dan Manajemen, [S.L.], v. 9, n. 2, p. 131-142, 14 dez. 2019. LP2M Universitas Islam Negeri (UIN) Syarif Hidayatullah Jakarta. http://dx.doi.org/10.15408/ess.v9i2.9724.

LU, Yuzhu; SMITH, Shana. Augmented Reality E-Commerce Assistant System: trying while shopping. **Lecture Notes In Computer Science**, [S.L.], p. 643-652, 2007. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-73107-8_72.

MEITU. **About Meitu**. Disponível em: https://www.meitu.com/en/introduction. Acesso em: 20 fev. 2022.

PACETE, Luiz Gustavo. Falta de informação motiva desistências no e-commerce. **Meio&mensagem**. 2019. Disponível em: https://www.meioemensagem.com.br/home/marketing/2019/09/05/falta-de-informacao-motiva-desistencias-no-e-commerce.html. Acesso em: 24 fev. 2022.

PERFECT CORP. **Benefit Cosmetics Brow Try-On Delivers “Incredible Realism”**

with YouCam Tech. 2022a. Disponível em: https://www.perfectcorp.com/business/successstory/detail/2. Acesso em: 20 fev. 2022.

PERFECT CORP. **E.l.f. Sees 200% Higher Conversion for Online Consumers Using Virtual Try-On**. 2022b. Disponível em: https://www.perfectcorp.com/business/successstory/detail/21. Acesso em: 20 fev. 2022.

PERFECT CORP. **Self-serve & Turnkey Virtual Try-on Solution for Ecommerce**. 2022c. Disponível em: https://www.perfectcorp.com/business/solutions/smb. Acesso em: 20 fev. 2022.

PERFECT CORP. **YouCam Makeup: Best Selfie Editor**. 2022d. Disponível em: https://www.perfectcorp.com/consumer/apps/ymk. Acesso em: 20 fev. 2022.

PEZZOTTI, Renato. Apple segue como marca mais valiosa do mundo; Tesla cresce 184%. **Uol economia**. 2021. Disponível em: https://economia.uol.com.br/noticias/redacao/2021/10/20/apple-segue-como-empresa-mais-valiosa-do-mundo-tesla-cresce-184.htm. Acesso em: 20 fev. 2022.

PTC. **PTC Announces a Major New Release to Vuforia Augmented Reality Platform**. 2017. Disponível em: https://www.ptc.com/en/news/2017/vuforia-7. Acesso em: 1 mar. 2022.

RIVEIRA, Carolina. Grupo Boticário compra site de e-commerce de cosméticos Beleza na Web. **Exame**. São Paulo, 2019. Disponível em: https://exame.com/negocios/grupo-boticario-compra-site-de-e-commerce-de-cosmeticos-beleza-na-web/. Acesso em: 20 fev. 2022.

SALEH, Khalid. E-commerce Product Return Rate – Statistics and Trends [Infographic]. **Invesp**. Disponível em:

https://www.invespcro.com/blog/ecommerce-product-return-rate-statistics/. Acesso em: 24 fev. 2022.

SAMPAIO, Daniel. O que é E-commerce? Tudo o que você precisa saber para ter uma loja virtual de sucesso! **Rocketcontent**. 2019. Disponível em: https://rockcontent.com/br/blog/e-commerce-guia/. Acesso em: 24 fev. 2022.

SAURO, Jeff. **Measuring Usability With The System Usability Scale (SUS)**. 2016. Disponível em: http://www.userfocus.co.uk/articles/measuring-usability-with-the-SUS.html. Acesso em: 20 jun. 2022.

SIZEBAY. **Prejuízos com trocas e devoluções no e-commerce fashion? Veja essas sugestões**. Disponível em: https://sizebay.com/pt/blog/prejuizos-com-trocas-e-devolucoes-no-e-commerce-fashion-veja-essas-sugestoes/. Acesso em: 24 fev. 2022.

SEPHORA. **About us**. Disponível em: https://www.sephora.com/beauty/about-us. Acesso em: 20 fev. 2022.

SILVA, S.; ABREU, A.; TEIXEIRA, A. P. Influência da Realidade Aumentada na Decisão de Compra dos Consumidores. **Research Bulletin (Cadernos de Investigação) of the Master in E-Business**, [S. l.], v. 1, n. 1, 2021. Disponível em: https://www.iscap.pt/ebusiness-rj/index.php/mne-rj/article/view/58. Acesso em: 6 mar. 2022.

VIVO, Patricio Gonzalez; LOWE, Jen. **The Book of Shaders**. 2015. Disponível em: https://thebookofshaders.com/01/. Acesso em: 18 jun. 2022.

VUFORIA. **Vuforia Engine Overview**. 2022a. Disponível em: https://library.vuforia.com/getting-started/vuforia-features. Acesso em: 1 mar. 2022.

VUFORIA. **Best method to recognize a face.** 2022b. Disponível em: https://developer.vuforia.com/forum/object-recognition/best-method-recognize-face. Acesso em: 11 de mar. de 2022.

WILLIAMS, Robert. Amazon rolls out AR lipstick try-ons via L'Oréal's ModiFace. **Marketing dive**. 2019. Disponível em: https://www.marketingdive.com/news/amazon-rolls-out-ar-lipstick-try-ons-via-lorealsmodiface/556202/. Acesso em: 20 fev. 2022.

YAOYUNEYONG, Gallayanee Starwind; POLLITTE, Wesley A.; FOSTER, Jamye K.; FLYNN, Leisa R.. Virtual dressing room media, buying intention and mediation. **Journal Of Research In Interactive Marketing**, [S.L.], v. 12, n. 1, p. 125-144, 16 fev. 2018. Emerald. http://dx.doi.org/10.1108/jrim-06-2017-0042.

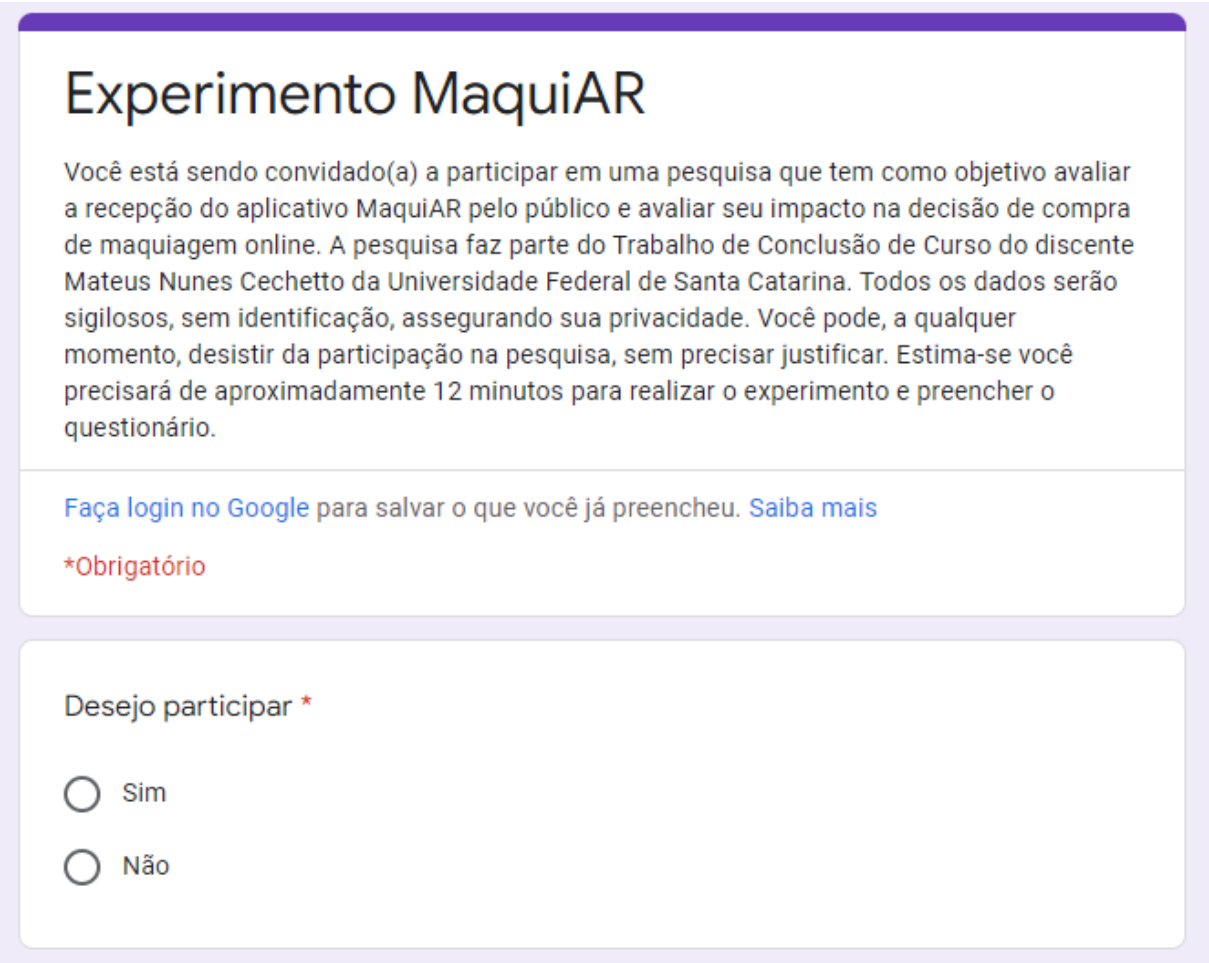
YIM, Mark Yi-Cheon; CHU, Shu-Chuan; SAUER, Paul L.. Is Augmented Reality Technology an Effective Tool for E-commerce? An Interactivity and Vividness Perspective. **Journal Of Interactive Marketing**, [S.L.], v. 39, p. 89-103, ago. 2017. Elsevier BV. http://dx.doi.org/10.1016/j.intmar.2017.04.001.

ZERO FRICTION FUTURE. **AI-ding discovery: How technology is transforming the purchase journey**. 2019. Disponível em: https://zerofrictionfuture.economist.com/articles/ai-ding-discovery-how-technology-is-transforming-the-purchase-journey/. Acesso em: 24 fev. 2022.

ZUINI, Priscila. Época Cosméticos fatura R$ 16 mi com produtos importados. **Exame**. São Paulo, 2013. Disponível em: https://exame.com/pme/epoca-cosmeticos-fatura-r-16-mi-com-produtos-importados/. Acesso em: 20 fev. 2022.

APÊNDICES

APÊNDICE A - Questionários do experimento



Experimento MaquiAR

Você está sendo convidado(a) a participar em uma pesquisa que tem como objetivo avaliar a recepção do aplicativo MaquiAR pelo público e avaliar seu impacto na decisão de compra de maquiagem online. A pesquisa faz parte do Trabalho de Conclusão de Curso do discente Mateus Nunes Cechetto da Universidade Federal de Santa Catarina. Todos os dados serão sigilosos, sem identificação, assegurando sua privacidade. Você pode, a qualquer momento, desistir da participação na pesquisa, sem precisar justificar. Estima-se você precisará de aproximadamente 12 minutos para realizar o experimento e preencher o questionário.

Faça login no Google para salvar o que você já preencheu. Saiba mais

*Obrigatório

Desejo participar *

- ◯ Sim
- ◯ Não

Questionário pré-uso do MaquiAR

As perguntas desta seção servem para avaliar sua familiaridade com compras de maquiagem online.

Qual a sua idade?

Sua resposta

Você costuma comprar maquiagem com frequência?

Sua resposta

Você já comprou maquiagem pela internet?

- ◯ Sim
- ◯ Não

Você já conhecia o produto antes de comprá-lo pela internet?

- ◯ Sim e comprei pela internet pois estava mais barato que na loja física
- ◯ Sim e comprei pela internet pela praticidade
- ◯ Não, comprei o produto pela primeira vez
- ◯ Nunca comprei maquiagem pela internet

Você já se arrependeu de comprar uma maquiagem pela internet?

- ◯ Sim
- ◯ Não

Qual o motivo de você ter se arrependido da compra?

Sua resposta

Você tem algum receio ao comprar maquiagem pela internet?

Sua resposta

Você já usou algum aplicativo de try-on de maquiagem? Que te permite experimentar virtualmente a maquiagem antes de comprá-la

- ◯ Sim e uso com frequência
- ◯ Sim mas não uso com frequência
- ◯ Não mas sabia que existiam
- ◯ Não e não sabia que existiam

Instruções para o uso do MaquiAR

Peça para a pessoa que está conduzindo a pesquisa para que lhe entregue o dispositivo com o MaquiAR instalado. O MaquiAR é um aplicativo de Realidade Aumentada que permite que o usuário experimente maquiagens antes de comprá-las. Neste experimento você pode deve experimentar ao menos 2 produtos diferentes, mas pode experimentar mais, se quiser! O condutor da pesquisa não lhe dará instruções de como o aplicativo funciona, a usabilidade do mesmo é um dos quesitos que será avaliado.

Questionário pós-uso do MaquiAR

As perguntas desta seção tem como objetivo avaliar a recepção do MaquiAR pelo usuário.

Como você avalia as maquiagens apresentadas no MaquiAR?

	1	2	3	4	5	
Muito pouco realistas	○	○	○	○	○	Muito realistas

Espaço para comentários sobre a resposta anterior

Sua resposta

Se um app como o MaquiAR estivesse disponível, você se sentiria mais confiante para realizar uma compra?

	1	2	3	4	5	
Menos confiante	○	○	○	○	○	Mais confiante

Espaço para comentários sobre a resposta anterior

Sua resposta

O que você acha da ideia de aplicativos de try-on de maquiagem?

	1	2	3	4	5	
Muito ruim	○	○	○	○	○	Muito boa

Espaço para comentários sobre a resposta anterior

Sua resposta

Em geral, como você avalia sua experiência utilizando o MaquiAR?

	1	2	3	4	5	
Muito ruim	○	○	○	○	○	Muito boa

O que achou do MaquiAR? Escreva o que quiser, sobre o aplicativo, sobre a ideia, etc.

Sua resposta

Questionário usabilidade do MaquiAR

Essa seção tem como objetivo avaliar a usabilidade do aplicativo

Avalie as seguintes afirmações

	Discordo totalmente	Discordo parcialmente	Nem concordo, nem discordo	Concordo parcialmente	Concordo totalmente
Eu penso que usarei este sistema com frequência	◯	◯	◯	◯	◯
Acho o sistema desnecessariamente complexo	◯	◯	◯	◯	◯
Penso que o sistema é fácil de usar	◯	◯	◯	◯	◯
Acho que vou precisar de ajuda de um técnico para usar este sistema	◯	◯	◯	◯	◯
Acho as funções deste sistema bem integradas	◯	◯	◯	◯	◯
Encontro muitas inconsistências neste sistema	◯	◯	◯	◯	◯
Imagino que as pessoas aprenderão rapidamente a usar este sistema	◯	◯	◯	◯	◯
Acho o sistema imprático de usar	◯	◯	◯	◯	◯
Senti-me confiante ao usar o sistema	◯	◯	◯	◯	◯
Precisei aprender muitas coisas antes de ser capaz de operar o sistema	◯	◯	◯	◯	◯

APÊNDICE B - Código fonte

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.maquiAR">

      <uses-permission  android:name="android.permission.CAMERA"
/>
                                                    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
android:maxSdkVersion="18"/>
                                                    <uses-permission
android:name="android.permission.INTERNET"/>

   <uses-feature
       android:name="android.hardware.camera.ar"
       android:required="true" />
   <uses-feature
       android:glEsVersion="0x00020000"
       android:required="true" />

   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:roundIcon="@mipmap/ic_launcher_round"
       android:supportsRtl="true"
       android:theme="@style/Theme.MaquiAR">
       <provider
           android:name="androidx.core.content.FileProvider"
           android:authorities="${applicationId}.provider"
```

```
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data

android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/provider_paths" />
        </provider>
        <activity
            android:name=".FavoriteActivity"
            android:exported="false" />
        <activity
            android:name=".ProductDetailActivity"
            android:exported="false" />
        <activity
            android:name=".ProductListActivity"
            android:exported="true">
            <intent-filter>
                                                    <action
android:name="android.intent.action.MAIN" />

                                                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ARVisualizationActivity"
            android:configChanges="orientation|screenSize"
            android:screenOrientation="locked">
            <intent-filter>
                                                    <action
android:name="android.intent.action.MAIN" />
```

```
                                                    <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="com.google.ar.core"
            android:value="required" />
    </application>

</manifest>

package com.maquiAR.Models

import com.maquiAR.R

class Blush(
    id: Int,
    name: String,
    description: String,
    images: ArrayList<String>,
    ecommerceLink: String,
    color: FloatArray,
    colorName: String
) : Product(
          id,   name,   description,   images,   ecommerceLink,
"models/blush.png",
    R.drawable.icon_blush, color, colorName
)

package com.maquiAR.Models
```

```
import com.maquiAR.R
import java.util.ArrayList

class EyeShadow(
    id: Int,
    name: String,
    description: String,
    images: ArrayList<String>,
    ecommerceLink: String,
    color: FloatArray,
    colorName: String
) : Product(id, name, description, images, ecommerceLink,
"models/eyeShadow.png", R.drawable.icon_eye_makeup, color,
colorName)

package com.maquiAR.Models

import com.maquiAR.R
import java.util.ArrayList

class Lipstick(
    id: Int,
    name: String,
    description: String,
    images: ArrayList<String>,
    ecommerceLink: String,
    color: FloatArray,
    colorName: String
) : Product(id, name, description, images, ecommerceLink,
"models/lipstick.png", R.drawable.icon_lipstick, color,
colorName)
```

```
package com.maquiAR.Models

import android.graphics.Color
import java.io.Serializable
import java.util.ArrayList
import kotlin.math.roundToInt

abstract class Product(
    val id: Int,
    val name: String,
    val description: String,
    val images: ArrayList<String>,
    val ecommerceLink: String,
    val texturePath: String,
    val icon: Int,
    val color: FloatArray,
    val colorName: String
) : Serializable

fun Product.getColorInt(): Int {
    val r = (color[0] * 255).roundToInt()
    val g = (color[1] * 255).roundToInt()
    val b = (color[2] * 255).roundToInt()
    val hex = String.format("#%02x%02x%02x", r, g, b)
    return Color.parseColor(hex)
}

package com.maquiAR.support

import android.app.Activity
import androidx.preference.PreferenceManager
import com.google.gson.Gson
```

```
import com.maquiAR.Models.Product

object FavoriteUtils {

   private var favoriteIds:  MutableList<Int> = ArrayList()


    private fun loadFavoriteIds(activity: Activity): List<Int>
{
                                        val     sharedPref     =
PreferenceManager.getDefaultSharedPreferences(activity)
         val json = sharedPref.getString("Favoritos", "[]") ?:
"[]"
       if(json.length < 3) {
           this.favoriteIds = ArrayList()
           return this.favoriteIds
       }
            var  stringList  =  json.substring(1,  json.length  -
1).split(",")
       var intList = stringList.map { string ->
           string.toInt()
       }.toMutableList()
       this.favoriteIds = intList
       return this.favoriteIds
   }

   private fun saveFavorites(activity: Activity) {
       val key = "Favoritos"
                                       val     sharedPref     =
PreferenceManager.getDefaultSharedPreferences(activity)
       val gson = Gson()
       val json: String = gson.toJson(favoriteIds)
```

```
        with(sharedPref.edit()) {
            putString(key, json)
            apply()
        }

    }

    fun getFavorites(activity: Activity): List<Int> {
        if(favoriteIds.isEmpty()) {
            loadFavoriteIds(activity)
        }
        return this.favoriteIds
    }

    fun addFavorite(id: Int, activity: Activity) {
        this.favoriteIds.add(id)
        saveFavorites(activity)
    }

    fun removeFavorite(id: Int, activity: Activity) {
        this.favoriteIds.remove(id)
        saveFavorites(activity)
    }

    fun isFavorite(id: Int) : Boolean {
        return favoriteIds.contains(id)
    }

         fun   getProductsByIdsList(allProducts:   List<Product>,
favoriteIds: List<Int>): List<Product> {
        return allProducts.filter { product ->
            favoriteIds.contains(product.id)
```

```
        }
    }
}

package com.maquiAR.support

import com.maquiAR.Models.Blush
import com.maquiAR.Models.EyeShadow
import com.maquiAR.Models.Lipstick
import com.maquiAR.Models.Product

object ProductLoader {

    private val allProducts: MutableList<Product> = ArrayList()

    fun loadProducts(): MutableList<Product> {
        if (allProducts.isEmpty()) {
            return allProducts
        }

        allProducts.add(
            Lipstick(
                1,
                "Batom e Brilho labial Luisance",
                "Marca: Luisance \n",
                ArrayList(
                    listOf(

"https://duvidasdebeleza.com.br/wp-content/uploads/2016/04/resenha-batom-liquido-matte-luisance-roxo512-blog-duvidasdebeleza-1024x925.jpg",
```

```
"https://www.oval.co.uk/wp-content/uploads/Pantone.5125C.jpg"
                    )
                ),

"https://www.formulacertaperfumaria.com.br/produto/batom-royal
-luisance/?attribute_pa_cor=03&utm_source=Google%20Shopping&ut
m_campaign=Google%20Shopping&utm_medium=organic&utm_term=13346
",
                floatArrayOf(0.467f, 0.173f, 0.251f, 1f),
                "Roxo 5125"
            )
        )


        val list = ArrayList<String>()

list.add("https://www.maccosmetics.com.br/media/export/cms/pro
ducts/640x600/mac_sku_M0N904_640x600_0.jpg")

list.add("https://www.maccosmetics.com.br/media/export/cms/pro
ducts/640x600/mac_sku_M0N904_640x600_3.jpg")

list.add("https://www.maccosmetics.com.br/media/export/cms/pro
ducts/smoosh_v2/mac_smoosh_M0N904.jpg")
        allProducts.add(
            Lipstick(
                2,
                "Batom Retro Matte",
                    "Marca: MAC \n M·A·C Lipstick - o produto
icônico que tornou a M·A·C famosa. Fórmula de longa duração
```

```
que proporciona cor intensamente pigmentada e acabamento ultra
matte aveludado.",
                list,

"https://www.maccosmetics.com.br/product/13854/52593/produtos/
maquiagem/labios/batom/batom-retro-matte?gclid=CjwKCAjwqauVBhB
GEiwAXOepkXAQ43pAIgIbdPv-cJVzrYp7kZTDJDtlKB3g_G2R58d6OkEA4VPP3
hoCMigQAvD_BwE&gclsrc=aw.ds#!/shade/ruby_woo",
                floatArrayOf(0.898f, 0.216f, 0.286f, 1f),
                "Ruby Woo"
            )
        )

        allProducts.add(
            Lipstick(
                3,
                "Batom",
                "Generico",
                ArrayList(
                    listOf(

"https://cdn-icons-png.flaticon.com/512/1024/1024505.png"
                    )
                ),

"https://www.google.com.br/search?tbm=shop&hl=pt-BR&&q=batom&o
q=batom",
                floatArrayOf(0.671f, 0.286f, 0.482f, 1f),
                "Roxo"
            )
        )
```

```
        allProducts.add(
            Lipstick(
                4,
                "Batom Matte HLLR Chic",
                  "Última moda em Paris, chega agora ao Brasil o
Batom Matte HL Chic™. Feito para durar o dia todo e deixar
seus lábios mais volumosos.",
                ArrayList(
                    listOf(

"https://cdn.shopify.com/s/files/1/0629/9582/4879/products/Des
ignsemnome_50_600x.jpg?v=1652683586",

"https://cdn.shopify.com/s/files/1/0629/9582/4879/products/Des
ignsemnome_46_0430a2ab-251f-4faf-a59b-b87126240123_600x.jpg?v=
1652683581"
                    )
                ),

"https://lojaferreirarodrigues.com.br/products/batom-matte-hll
r-chic-paris?variant=42804346355951&currency=BRL&utm_medium=pr
oduct_sync&utm_source=google&utm_content=sag_organic&utm_campa
ign=sag_organic",
                floatArrayOf(0.639f, 0.208f, 0.212f, 1f),
                "Vermelho Paixão"
            )
        )




        allProducts.add(
```

```
            EyeShadow(
                5,
                "Sombra de olho",
                " ",

ArrayList(listOf("https://cdn-icons-png.flaticon.com/512/3557/
3557187.png")),

"https://www.google.com.br/search?tbm=shop&hl=pt-BR&&q=sombra+
de+olho&oq=sombra+de+olho",
                floatArrayOf(0.647f, 0.357f, 0.33f, 1f),

                "nude 1"
            )
        )

        allProducts.add(
            EyeShadow(
                6,
                "Sombra de olho",
                " ",

ArrayList(listOf("https://cdn-icons-png.flaticon.com/512/3557/
3557187.png")),

"https://www.google.com.br/search?tbm=shop&hl=pt-BR&&q=sombra+
de+olho&oq=sombra+de+olho",
                floatArrayOf(0.706f, 0.435f, 0.314f, 1f),

                "nude 2"
            )
        )
```

```
        allProducts.add(
            Blush(
                7,
                "Blush",
                " ",

ArrayList(listOf("https://cdn-icons-png.flaticon.com/512/1024/
1024537.png")),

"https://www.google.com.br/search?tbm=shop&hl=pt-BR&&q=blush&o
q=blush",
                floatArrayOf(0.706f, 0.435f, 0.314f, 1f),
                "nude 2"
            )
        )

        allProducts.add(
            Blush(
                7,
                "Blush",
                " ",

ArrayList(listOf("https://cdn-icons-png.flaticon.com/512/1024/
1024537.png")),

"https://www.google.com.br/search?tbm=shop&hl=pt-BR&&q=blush&o
q=blush",
                floatArrayOf(0.647f, 0.357f, 0.33f, 1f),
                "nude 1"
            )
```

```
        )


        return allProducts
    }

    fun getRandomProduct(): Product {
        return allProducts.random()
    }
}
```

```
package com.maquiAR

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.ImageButton
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.maquiAR.Models.Product
import com.maquiAR.arface.AugmentedFaceFragment
import com.maquiAR.arface.AugmentedFaceListener
import com.maquiAR.arface.AugmentedFaceNode
import com.maquiAR.support.FavoriteUtils
import com.maquiAR.support.ProductLoader
import
com.maquiAR.support.recyclers.ARVisualizationProductCardViewRe
cycler
```

```
import
kotlinx.android.synthetic.main.activity_ar_visualization.*


class     ARVisualizationActivity     :     AppCompatActivity(),
AugmentedFaceListener,

ARVisualizationProductCardViewRecycler.OnProductSelectedListen
er {

   private var productChanged = false
   private lateinit var productPicked: Product

   private lateinit var openDescriptionButton: ImageButton
   private lateinit var addToFavoriteButton: ImageButton
   private lateinit var takePhotoButton: ImageButton
   private lateinit var listProductsButton: ImageButton
   private lateinit var listProductsRecycler: RecyclerView
   private lateinit var toolbar: Toolbar

           private    lateinit    var    augmentedFaceFragment:
AugmentedFaceFragment

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_ar_visualization)

                                          productPicked      =
intent.extras!!.getSerializable("Product") as Product
       loadToolbar()
       loadLayoutElements()
```

```
                augmentedFaceFragment = face_view as
AugmentedFaceFragment
        augmentedFaceFragment.setAugmentedFaceListener(this)
    }

    override fun onFaceAdded(face: AugmentedFaceNode) {
        face.setFaceMeshTexture(productPicked.texturePath)
        face.setTextureColor(productPicked.color)
    }

    override fun onFaceUpdate(face: AugmentedFaceNode) {
        if (productChanged) {
            productChanged = false
            face.setFaceMeshTexture(productPicked.texturePath)
            face.setTextureColor(productPicked.color)
        }
    }

    private fun loadToolbar() {
        toolbar = findViewById(R.id.app_toolbar)
              toolbar.title = productPicked.name + " - " +
productPicked.colorName
        setSupportActionBar(toolbar)
        supportActionBar?.apply{
            setDisplayHomeAsUpEnabled(false)
            setDisplayShowHomeEnabled(false)
        }
    }

    private fun loadLayoutElements() {
                                    openDescriptionButton =
findViewById(R.id.openProductDescriptionButton)
```

```
openDescriptionButton.setImageResource(R.drawable.icon_cart)
        addToFavoriteButton = findViewById(R.id.toggleFavorite)
        setFavoriteIcon()
        takePhotoButton = findViewById(R.id.takePhotoButton)

takePhotoButton.setImageResource(R.drawable.icon_camera)

        listProductsButton = findViewById(R.id.listProducts)

listProductsButton.setImageResource(R.drawable.icon_list)

        loadProducts()
    }

    private fun loadProducts() {
        loadProductsRecycler(ProductLoader.loadProducts())
    }

    private fun loadProductsRecycler(products: List<Product>) {
                                    val     prodCardAdapter     =
ARVisualizationProductCardViewRecycler(this, products, this)
                                        listProductsRecycler     =
findViewById<RecyclerView>(R.id.recycler_simple_products_list)
.apply {
            visibility = View.VISIBLE
            layoutManager =

LinearLayoutManager(this@ARVisualizationActivity,
LinearLayoutManager.HORIZONTAL, false)
            adapter = prodCardAdapter
        }
```

```
    }

    override fun onProductSelected(product: Product) {
        this.productPicked = product
              toolbar.title = productPicked.name + " - " +
productPicked.colorName
        setFavoriteIcon()
        productChanged = true

    }

    fun openProductDescription(view: View?) {
                            val intent = Intent(this,
ProductDetailActivity::class.java)
        intent.putExtra("ProductCheck", productPicked)
        startActivity(intent)
    }

    fun toggleFavorite(view: View?) {
        if (checkProductInFavorite()) {
            removeProductFromFavorite()
        } else {
            addProductToFavorite()
        }
    }

    fun checkProductInFavorite(): Boolean {
        return FavoriteUtils.isFavorite(this.productPicked.id)
    }

    fun setFavoriteIcon() {
```

```
        if (checkProductInFavorite()) {

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
64dp)
        } else {

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
bordered_64dp)
        }
    }

    fun addProductToFavorite() {
        FavoriteUtils.addFavorite(productPicked.id, this)
        Toast.makeText(
                    applicationContext, "Produto adicionado aos
favoritos!",
            Toast.LENGTH_SHORT
        ).show()

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
64dp)
    }

    fun removeProductFromFavorite() {
        FavoriteUtils.removeFavorite(productPicked!!.id, this)
        Toast.makeText(
                    applicationContext, "Produto removido dos
favoritos!",
            Toast.LENGTH_SHORT
        ).show()
```

```
addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
bordered_64dp)
    }

    fun takePhoto(view: View) {
        augmentedFaceFragment.takeScreenshot = true
    }

    fun toggleProductList(view: View) {
          if (listProductsRecycler.visibility == View.INVISIBLE)
{
            listProductsRecycler.visibility = View.VISIBLE
        } else {
            listProductsRecycler.visibility = View.INVISIBLE
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return super.onSupportNavigateUp()
    }


}

package com.maquiAR

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
```

```
import android.widget.TextView
import androidx.appcompat.widget.Toolbar
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView
import
com.google.android.material.bottomnavigation.BottomNavigationV
iew
import com.maquiAR.Models.Product
import com.maquiAR.support.recyclers.ProductCardViewRecycler
import com.maquiAR.support.FavoriteUtils
import com.maquiAR.support.ProductLoader

class FavoriteActivity : AppCompatActivity() {

    private var allProducts: List<Product> = ArrayList()
    private var favoriteIds:  List<Int> = ArrayList()
    private var favoriteProducts: List<Product> = ArrayList()


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_product_list)

        loadToolbar()
        loadBottomMenuNavigation()

        allProducts = ProductLoader.loadProducts()
        favoriteIds = FavoriteUtils.getFavorites(this)
                                        favoriteProducts      =
FavoriteUtils.getProductsByIdsList(allProducts, favoriteIds)

        showProducts(favoriteProducts)
```

```
    }


    private fun showProducts(products: List<Product>) {
                                    val     productExhibition     =
findViewById<View>(R.id.recycler_products_list)              as
RecyclerView
            val  prodCardAdapter  =  ProductCardViewRecycler(this,
products)
                                productExhibition.layoutManager    =
GridLayoutManager(this, 2)
        productExhibition.adapter = prodCardAdapter

         val  emptyText  =  findViewById<View>(R.id.empty_view)  as
TextView
        if(products.isEmpty()) {
            productExhibition.visibility = View.GONE
            emptyText.visibility = View.VISIBLE
        } else {
            productExhibition.visibility = View.VISIBLE
            emptyText.visibility = View.GONE
        }
    }

    private fun loadToolbar() {
          val  toolbar  =  findViewById<View>(R.id.app_toolbar)  as
Toolbar
        toolbar.title = "Favoritos"
        setSupportActionBar(toolbar)
        supportActionBar?.apply{
            setDisplayHomeAsUpEnabled(false)
            setDisplayShowHomeEnabled(false)
```

```
        }
    }

    private fun loadBottomMenuNavigation() {
                                        val      bottomMenu      =
findViewById<BottomNavigationView>(R.id.bottom_menu)

bottomMenu.setOnNavigationItemSelectedListener(listener)
        bottomMenu.selectedItemId = R.id.menuFavorite
    }

    var listener =
         BottomNavigationView.OnNavigationItemSelectedListener {
item ->
            val intent: Intent
            when (item.itemId) {
                R.id.menuCamera -> {
                            intent = Intent(this@FavoriteActivity,
ARVisualizationActivity::class.java)
                                        intent.putExtra("Product",
ProductLoader.getRandomProduct())
                    startActivity(intent)
                }
                R.id.menuProducts -> {
                            intent = Intent(this@FavoriteActivity,
ProductListActivity::class.java)
                    startActivity(intent)
                }
                R.id.menuFavorite -> {}
            }
            true
        }
```

```
}

package com.maquiAR

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.ImageButton
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import com.denzcoskun.imageslider.ImageSlider
import com.denzcoskun.imageslider.models.SlideModel
import
com.google.android.material.bottomnavigation.BottomNavigationV
iew
import com.maquiAR.Models.Product
import com.maquiAR.support.FavoriteUtils

class ProductDetailActivity : AppCompatActivity() {

    private lateinit var productPicked: Product

    //Layout TextView fields
    private lateinit var prodName: TextView
    private lateinit var prodDescription: TextView
    private lateinit var prodColor: TextView
```

```
    private lateinit var prodARViewButton: Button
    private lateinit var addToFavoriteButton: ImageButton

    private lateinit var imageSlider: ImageSlider
    private val imageList = ArrayList<SlideModel>()


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_product_detail)

        loadToolbar()
        loadBottomMenuNavigation()
        getLayoutElements()

        val intent = intent
                                            productPicked      =
intent.extras!!.getSerializable("ProductCheck") as Product

        loadImageSlider()

        if (checkProductInFavorite()) {

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
64dp)
        } else {

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
bordered_64dp)
        }
```

```
        setProductInfo()

    }

    private fun loadToolbar() {
          val toolbar = findViewById<View>(R.id.app_toolbar) as
Toolbar
        toolbar.title = "Sobre o Produto"
        setSupportActionBar(toolbar)
        supportActionBar?.apply{
            setDisplayHomeAsUpEnabled(false)
            setDisplayShowHomeEnabled(false)
        }
    }

    private fun loadBottomMenuNavigation() {
                                        val     bottomMenu     =
findViewById<BottomNavigationView>(R.id.bottom_menu)
        bottomMenu.selectedItemId = R.id.menuProducts

bottomMenu.setOnNavigationItemSelectedListener(listener)
    }

    private var listener =
         BottomNavigationView.OnNavigationItemSelectedListener {
item ->
            val intent: Intent
            when (item.itemId) {
                R.id.menuCamera -> {
                     intent = Intent(this@ProductDetailActivity,
ARVisualizationActivity::class.java)
                    intent.putExtra("Product", productPicked)
```

```
                    startActivity(intent)
                }
                R.id.menuProducts -> {
                     intent = Intent(this@ProductDetailActivity,
ProductListActivity::class.java)
                    startActivity(intent)
                }
                R.id.menuFavorite -> {
                     intent = Intent(this@ProductDetailActivity,
FavoriteActivity::class.java)
                    startActivity(intent)
                }
            }
            true
        }

    fun getLayoutElements() {
                                        prodARViewButton      =
findViewById<View>(R.id.ar_visualization_button) as Button
                                    addToFavoriteButton      =
findViewById<View>(R.id.toggleFavorite) as ImageButton
              prodName  =  findViewById<View>(R.id.prodName)  as
TextView
                                        prodDescription      =
findViewById<View>(R.id.prodTextDescription) as TextView
             prodColor  =  findViewById<View>(R.id.prodColor)  as
TextView
    }

    private fun loadImageSlider() {
        for (i in productPicked.images) {
            imageList.add(SlideModel(i))
```

```
        }
                                                imageSlider      =
findViewById<ImageSlider>(R.id.image_slider)
        imageSlider.setImageList(imageList)

    }

    private fun checkProductInFavorite(): Boolean {
        return FavoriteUtils.isFavorite(this.productPicked.id)
    }

    private fun setProductInfo() {
        productPicked.apply {
            prodName.text = name
            prodColor.text = "Cor: $colorName"
            prodDescription.text = description
        }
    }

    fun addProductToFavorite() {
        FavoriteUtils.addFavorite(productPicked.id, this)
        Toast.makeText(
                    applicationContext, "Produto adicionado aos
favoritos!",
            Toast.LENGTH_SHORT
        ).show()

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
64dp)
    }

    fun removeProductFromFavorite() {
```

```
        FavoriteUtils.removeFavorite(productPicked.id, this)
        Toast.makeText(
                    applicationContext, "Produto removido dos
favoritos!",
            Toast.LENGTH_SHORT
        ).show()

addToFavoriteButton.setImageResource(R.drawable.icon_favorite_
bordered_64dp)
    }

    fun checkProductAR(view: View) {
                                val intent = Intent(this,
ARVisualizationActivity::class.java)
        intent.putExtra("Product", productPicked)
        this.startActivity(intent)
    }

    fun goToWebsite(view: View?) {
        val intent = Intent(Intent.ACTION_VIEW)
        intent.setData(Uri.parse(productPicked.ecommerceLink))
        startActivity(intent)
    }

    fun toggleFavorite(view: View?) {
        if (checkProductInFavorite()) {
            removeProductFromFavorite()
        } else {
            addProductToFavorite()
        }
    }
```

```
    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return super.onSupportNavigateUp()
    }


}

package com.maquiAR

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.RecyclerView
import
com.google.android.material.bottomnavigation.BottomNavigationV
iew
import com.maquiAR.Models.Product
import com.maquiAR.support.recyclers.ProductCardViewRecycler
import com.maquiAR.support.FavoriteUtils
import com.maquiAR.support.ProductLoader

class ProductListActivity : AppCompatActivity() {

    private var allProducts: MutableList<Product> = ArrayList()


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_product_list)

        loadToolbar()
        loadBottomMenuNavigation()

        allProducts = ProductLoader.loadProducts()

        FavoriteUtils.getFavorites(this)

        showProducts(allProducts)

    }

    private fun showProducts(products: List<Product>) {
                                    val     productExhibition     =
findViewById<View>(R.id.recycler_products_list)                 as
RecyclerView
            val  prodCardAdapter  =  ProductCardViewRecycler(this,
products)
                                productExhibition.layoutManager    =
GridLayoutManager(this, 2)
        productExhibition.adapter = prodCardAdapter
    }

    private fun loadToolbar() {
          val  toolbar  =  findViewById<View>(R.id.app_toolbar)  as
Toolbar
        toolbar.title = "Todas as Maquiagens"
        setSupportActionBar(toolbar)
        supportActionBar?.apply{
            setDisplayHomeAsUpEnabled(false)
            setDisplayShowHomeEnabled(false)
```

```
        }
    }

    private fun loadBottomMenuNavigation() {
                                        val      bottomMenu      =
findViewById<BottomNavigationView>(R.id.bottom_menu)

bottomMenu.setOnNavigationItemSelectedListener(listener)
        bottomMenu.selectedItemId = R.id.menuProducts
    }

    private var listener =
         BottomNavigationView.OnNavigationItemSelectedListener {
item ->
            val intent: Intent
            when (item.itemId) {
                R.id.menuCamera -> {
                        intent = Intent(this@ProductListActivity,
ARVisualizationActivity::class.java)
                                        intent.putExtra("Product",
ProductLoader.getRandomProduct())
                    startActivity(intent)
                }
                R.id.menuProducts -> {
                }
                R.id.menuFavorite -> {
                        intent = Intent(this@ProductListActivity,
FavoriteActivity::class.java )
                    startActivity(intent)
                }
            }
            true
```

```
        }

    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return super.onSupportNavigateUp()
    }
}

package com.maquiAR.support.recyclers

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
import android.widget.TextView
import androidx.cardview.widget.CardView
import androidx.recyclerview.widget.RecyclerView
import com.maquiAR.Models.Product
import com.maquiAR.Models.getColorInt
import com.maquiAR.R

class ARVisualizationProductCardViewRecycler(
    private val context: Context,
    private val productListData: List<Product>,
                private     val     onProductSelectedListener:
OnProductSelectedListener
) :

RecyclerView.Adapter<ARVisualizationProductCardViewRecycler.Pr
oductView>() {
```

```
        override fun onCreateViewHolder(parent: ViewGroup,
viewType: Int): ProductView {
        val view: View
        val mInflater = LayoutInflater.from(context)
                                                    view =
mInflater.inflate(R.layout.ar_visualization_card_item_product_
view, parent, false)
        return ProductView(view)
    }

        override fun onBindViewHolder(holder: ProductView,
position: Int) {

                                    holder.productName.text =
productListData[position].name

holder.productColor.setBackgroundColor(productListData[positio
n].getColorInt())

holder.productImage.setImageResource(productListData[position]
.icon)
        holder.productCardView.setOnClickListener {

onProductSelectedListener.onProductSelected(productListData[po
sition])
        }
    }

    override fun getItemCount(): Int {
        return productListData.size
    }
```

```
            class     ProductView(productItem:     View)     :
RecyclerView.ViewHolder(productItem) {
        val productCardView: CardView
        val productImage: ImageView
        val productName: TextView
        val productColor: TextView

        init {
                                                productCardView    =
productItem.findViewById<CardView>(R.id.card_product_view)
                                                  productImage     =
productItem.findViewById<ImageView>(R.id.product_card_image)
                                                   productName     =
productItem.findViewById<TextView>(R.id.product_card_name)
                                                  productColor     =
productItem.findViewById<TextView>(R.id.product_card_color)
        }
    }

    interface OnProductSelectedListener {
        fun onProductSelected(product: Product)
    }
}

package com.maquiAR.support.recyclers

import android.content.Context
import android.content.Intent
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageView
```

```
import android.widget.TextView
import androidx.cardview.widget.CardView
import androidx.recyclerview.widget.RecyclerView
import com.maquiAR.ARVisualizationActivity
import com.maquiAR.Models.Product
import com.maquiAR.Models.getColorInt
import com.maquiAR.R

class ProductCardViewRecycler(
   private val context: Context,
   private val productListData: List<Product>) :
    RecyclerView.Adapter<ProductCardViewRecycler.ProductView>()
{

        override fun onCreateViewHolder(parent: ViewGroup,
viewType: Int): ProductView {
       val view: View
       val mInflater = LayoutInflater.from(context)
                                                  view =
mInflater.inflate(R.layout.card_item_product_view, parent,
false)
       return ProductView(view)
   }

        override fun onBindViewHolder(holder: ProductView,
position: Int) {

                                    holder.productName.text =
productListData[position].name

holder.productDescription.setBackgroundColor(productListData[p
osition].getColorInt())
```

```
holder.productImage.setImageResource(productListData[position]
.icon)
        holder.productCardView.setOnClickListener {
                                val   intent   =   Intent(context,
ARVisualizationActivity::class.java)
                                        intent.putExtra("Product",
productListData[position])
            context.startActivity(intent)
        }
    }

    override fun getItemCount(): Int {
        return productListData.size
    }

                class     ProductView(productItem:     View)     :
RecyclerView.ViewHolder(productItem) {
        val productCardView: CardView
        val productImage: ImageView
        val productName: TextView
        val productDescription: TextView

        init {
                                                productCardView    =
productItem.findViewById<CardView>(R.id.card_product_view)
                                                  productImage     =
productItem.findViewById<ImageView>(R.id.product_card_image)
                                                   productName     =
productItem.findViewById<TextView>(R.id.product_card_name)
            productDescription =
```

```
productItem.findViewById<TextView>(R.id.product_card_descripti
on)
        }
    }

}

package com.maquiAR.arface

import android.Manifest
import android.app.AlertDialog
import android.content.Intent
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.Bitmap.CompressFormat
import android.net.Uri
import android.opengl.GLES20
import android.opengl.GLSurfaceView
import android.os.Bundle
import android.os.Environment
import android.provider.Settings
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.FrameLayout
import androidx.core.app.ActivityCompat.checkSelfPermission
import androidx.core.content.ContextCompat
import androidx.core.content.FileProvider
import androidx.fragment.app.Fragment
import com.google.ar.core.*
import com.google.ar.core.ArCoreApk.InstallStatus
```

```
import com.google.ar.core.Session.Feature
import com.google.ar.core.exceptions.*
import com.maquiAR.R
import com.maquiAR.arface.helpers.DisplayRotationHelper
import com.maquiAR.arface.helpers.SnackbarHelper
import com.maquiAR.arface.helpers.TrackingStateHelper
import com.maquiAR.arface.rendering.BackgroundRenderer
import java.io.File
import java.io.FileOutputStream
import java.io.IOException
import java.nio.IntBuffer
import java.text.SimpleDateFormat
import java.util.*
import javax.microedition.khronos.egl.EGLConfig
import javax.microedition.khronos.opengles.GL10


public     class     AugmentedFaceFragment     :     Fragment(),
GLSurfaceView.Renderer {

   private var session: Session? = null

   private var frameLayout: FrameLayout? = null
   private var surfaceView: GLSurfaceView? = null
            private    lateinit    var    displayRotationHelper:
DisplayRotationHelper
             private    lateinit    var    trackingStateHelper:
TrackingStateHelper

             var     faceNodeMap     =     HashMap<AugmentedFace,
AugmentedFaceNode>()
```

```
        private val backgroundRenderer: BackgroundRenderer =
BackgroundRenderer()

    private var installRequested = false
    private var canRequestDangerousPermissions = true
        private val messageSnackbarHelper: SnackbarHelper =
SnackbarHelper()
    private val RC_PERMISSIONS = 1010

     private var augmentedFaceListener: AugmentedFaceListener? =
null

    var takeScreenshot: Boolean = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        displayRotationHelper = DisplayRotationHelper(context)
                                        trackingStateHelper     =
TrackingStateHelper(requireActivity())
        installRequested = false
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        frameLayout =
              inflater.inflate(R.layout.fragment_augmented_face,
container, false) as FrameLayout
```

```
                                                surfaceView       =
frameLayout?.findViewById<View>(R.id.surface_view)            as
GLSurfaceView
        surfaceView?.let {
            it.preserveEGLContextOnPause = true
            it.setEGLContextClientVersion(2)
             it.setEGLConfigChooser(8, 8, 8, 8, 16, 0) // Alpha
used for plane blending.

            it.setRenderer(this)
                                                it.renderMode     =
GLSurfaceView.RENDERMODE_CONTINUOUSLY
            it.setWillNotDraw(false)
        }

        return frameLayout
    }

    override fun onResume() {
        super.onResume()
        if (session == null) {
            var exception: Exception? = null
            var message: String? = null
            try {
                val installStatus = ArCoreApk.getInstance()

.requestInstall(requireActivity(), !installRequested)
                when (installStatus) {
                    InstallStatus.INSTALL_REQUESTED -> {
                        installRequested = true
                        return
                    }
```

```
                    InstallStatus.INSTALLED -> {
                    }
                    else -> {
                        println("Undefined installed status")
                    }
                }
                    // ARCore requires camera permissions to
operate. If we did not yet obtain runtime
                // permission on Android M and above, now is a
good time to ask the user for it.
                                                        if
(ContextCompat.checkSelfPermission(requireActivity(),
"android.permission.CAMERA")
                    == PackageManager.PERMISSION_GRANTED) {
                        // Configure session to use front facing
camera.
                    val featureSet: EnumSet<Feature> =
                        EnumSet.of(Feature.FRONT_CAMERA)
                    // Create the session.
                      session = Session( /* context= */context,
featureSet)
                    configureSession()
                } else {
                    requestDangerousPermissions()
                }

            } catch (e: UnavailableArcoreNotInstalledException)
{
                message = "Please install ARCore"
                exception = e
                                                } catch (e:
UnavailableUserDeclinedInstallationException) {
```

```
                message = "Please install ARCore"
                exception = e
            } catch (e: UnavailableApkTooOldException) {
                message = "Please update ARCore"
                exception = e
            } catch (e: UnavailableSdkTooOldException) {
                message = "Please update this app"
                exception = e
                                                } catch (e:
UnavailableDeviceNotCompatibleException) {
                message = "This device does not support AR"
                exception = e
            } catch (e: Exception) {
                message = "Failed to create AR session"
                exception = e
            }
            if (message != null) {

messageSnackbarHelper.showError(requireActivity(), message)
                                    println("Exception  creating
session:$exception")
                return
            }
        }
         // Note that order matters - see the note in onPause(),
the reverse applies here.

        try {
            session?.resume()
        } catch (e: CameraNotAvailableException) {
            messageSnackbarHelper.showError(
                requireActivity(),
```

```
                "Camera not available. Try restarting the app."
            )
            session = null
            return
        }

        surfaceView?.onResume()
        displayRotationHelper.onResume()
    }

    fun requestDangerousPermissions() {
        if (!canRequestDangerousPermissions) {
            // If this is in progress, don't do it again.
            return
        }
        canRequestDangerousPermissions = false

        val permissions: ArrayList<String> = ArrayList()
               val  additionalPermissions:  ArrayList<String>  =
ArrayList()
        val permissionLength = additionalPermissions.size
        for (i in 0 until permissionLength) {
                    if  (checkSelfPermission(requireActivity(),
additionalPermissions[i])
                != PackageManager.PERMISSION_GRANTED
            ) {
                permissions.add(additionalPermissions[i])
            }
        }

        // Always check for camera permission
```

```
                    if   (checkSelfPermission(requireActivity(),
Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED
        ) {
            permissions.add(Manifest.permission.CAMERA)
        }

        if (!permissions.isEmpty()) {
            // Request the permissions
            requestPermissions(

permissions.toArray(arrayOfNulls<String>(permissions.size)),
                RC_PERMISSIONS
            )
        }
    }

             public    fun    setAugmentedFaceListener(listener:
AugmentedFaceListener) {
        augmentedFaceListener = listener
    }
    /**
     * If true, [.requestDangerousPermissions] returns without
doing anything, if false
     * permissions will be requested
     */
    private fun getCanRequestDangerousPermissions(): Boolean? {
        return canRequestDangerousPermissions
    }

    /**
```

```
     * If true, [.requestDangerousPermissions] returns without
doing anything, if false
     * permissions will be requested
     */
                                          private              fun
setCanRequestDangerousPermissions(canRequestDangerousPermissio
ns: Boolean?) {
                          this.canRequestDangerousPermissions    =
canRequestDangerousPermissions!!
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        if (checkSelfPermission(
                requireActivity(),
                Manifest.permission.CAMERA
            )
            == PackageManager.PERMISSION_GRANTED
        ) {
            return
        }
        val builder: AlertDialog.Builder = AlertDialog.Builder(

requireActivity(),

android.R.style.Theme_Material_Dialog_Alert
                                                )
        builder
            .setTitle("Camera permission required")
```

```
            .setMessage("Add camera permission via Settings?")
              .setPositiveButton(android.R.string.ok) { dialog,
which ->
                // If Ok was hit, bring up the Settings app.
                val intent = Intent()
                                                intent.action   =
Settings.ACTION_APPLICATION_DETAILS_SETTINGS
                intent.data = Uri.fromParts(
                    "package",
                    requireActivity().packageName,
                    null
                )
                requireActivity().startActivity(intent)
                 // When the user closes the Settings app, allow
the app to resume.
                  // Allow the app to ask for permissions again
now.
                setCanRequestDangerousPermissions(true)
            }
            .setNegativeButton(android.R.string.cancel, null)
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setOnDismissListener {
                 // canRequestDangerousPermissions will be true
if "OK" was selected from the dialog,
                   // false otherwise.  If "OK" was selected do
nothing on dismiss, the app will
                // continue and may ask for permission again if
needed.
                     // If anything else happened, finish the
activity when this dialog is
                // dismissed.
                if (!getCanRequestDangerousPermissions()!!) {
```

```
                    requireActivity().finish()
                }
            }
            .show()
    }

    override fun onPause() {
        super.onPause()
        if (session != null) {
            // Note that the order matters - GLSurfaceView is
paused first so that it does not try
            // to query the session. If Session is paused
before GLSurfaceView, GLSurfaceView may
            // still call session.update() and get a
SessionPausedException.
            displayRotationHelper.onPause()
            surfaceView?.onPause()
            session?.pause()
        }
    }

    override fun onDestroy() {
        if (session != null) {
            // Explicitly close ARCore Session to release
native resources.
            // Review the API reference for important
considerations before calling close() in apps with
            // more complicated lifecycle requirements:
                                                        //
https://developers.google.com/ar/reference/java/arcore/referen
ce/com/google/ar/core/Session#close()
            session?.close()
```

```
            session = null
        }
        super.onDestroy()
    }

    override fun onDrawFrame(gl: GL10?) {
        // Clear screen to notify driver it should not load any
pixels from previous frame.
        // Clear screen to notify driver it should not load any
pixels from previous frame.
                GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT  or
GLES20.GL_DEPTH_BUFFER_BIT)

        session?.let {
            // Notify ARCore session that the view size changed
so that the perspective matrix and
            // the video background can be properly adjusted.
            // Notify ARCore session that the view size changed
so that the perspective matrix and
            // the video background can be properly adjusted.
            displayRotationHelper.updateSessionIfNeeded(it)

            try {

it.setCameraTextureName(backgroundRenderer.textureId)

                // Obtain the current frame from ARSession.
When the configuration is set to
                // UpdateMode.BLOCKING (it is by default), this
will throttle the rendering to the
                // camera framerate.
                val frame: Frame = it.update()
```

```
                val camera: Camera = frame.camera

                // Get projection matrix.
                val projectionMatrix = FloatArray(16)
                 camera.getProjectionMatrix(projectionMatrix, 0,
0.1f, 100.0f)

                // Get camera matrix and draw.
                val viewMatrix = FloatArray(16)
                camera.getViewMatrix(viewMatrix, 0)

                  // Compute lighting from average intensity of
the image.
                 // The first three components are color scaling
factors.
                  // The last one is the average pixel intensity
in gamma space.
                val colorCorrectionRgba = FloatArray(4)

frame.lightEstimate.getColorCorrection(colorCorrectionRgba, 0)

                   // If frame is ready, render camera preview
image to the GL surface.
                backgroundRenderer.draw(frame)

                 // Keep the screen unlocked while tracking, but
allow it to lock when tracking stops.

trackingStateHelper.updateKeepScreenOnFlag(camera.trackingStat
e)
                val faces: Collection<AugmentedFace> =
```

```
it.getAllTrackables(AugmentedFace::class.java)
                for (face in faces) {
                    if (!faceNodeMap.containsKey(face)) {
                          val faceNode = AugmentedFaceNode(face,
requireContext())

augmentedFaceListener?.onFaceAdded(faceNode)
                        faceNodeMap[face] = faceNode
                    } else {
                        faceNodeMap[face]?.let{ node ->

augmentedFaceListener?.onFaceUpdate(node)
                        }
                    }

                    val iter = faceNodeMap.entries.iterator()
                    while (iter.hasNext()) {
                        val entry = iter.next()
                        val faceNode = entry.key
                                  if (faceNode.trackingState ==
TrackingState.STOPPED) {
                            iter.remove()
                        }
                    }
                                     if (face.trackingState !==
TrackingState.TRACKING) {
                        break
                    }
                    // Face objects use transparency so they
must be rendered back to front without depth write.
                    GLES20.glDepthMask(false)
```

```
                    // Each face's region poses, mesh vertices,
and mesh normals are updated every frame.

                     // 1. Render the face mesh first, behind
any 3D objects attached to the face regions.
                    faceNodeMap[face]?.onDraw(projectionMatrix,
viewMatrix, colorCorrectionRgba)
                }
            } catch (t: Throwable) {
                    // Avoid crashing the application due to
unhandled exceptions.
                println("Exception on the OpenGL thread $t")
            } finally {
                GLES20.glDepthMask(true)
                if(takeScreenshot) {
                    shareImage(saveBitmap(takeScreenshot(gl)))
                    takeScreenshot = false
                }
            }
        }
    }

    fun shareImage(uri: Uri?) {
        val intent = Intent(Intent.ACTION_SEND)
        intent.putExtra(Intent.EXTRA_STREAM, uri)
        intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
            intent.putExtra(Intent.EXTRA_TEXT, "O que achou da
minha maquiagem? - Filtro criado com MaquiAR")
        intent.type = "image/png"
        startActivity(Intent.createChooser(intent, null))
```

```
    }

    fun saveBitmap(bitmap: Bitmap?): Uri? {
        var uri: Uri? = null

        try {
            val dateFormatter by lazy {
                SimpleDateFormat(
                                "yyyy.MM.dd 'at' HH:mm:ss z",
Locale.getDefault()
                )
            }
                                                    val    file    =
File(requireContext().getExternalFilesDir(Environment.DIRECTOR
Y_PICTURES), "MaquiAR${dateFormatter.format(Date())}.png")
            val stream = FileOutputStream(file)
            bitmap!!.compress(CompressFormat.PNG, 100, stream)
            stream.flush()
            stream.close()
              uri = FileProvider.getUriForFile(requireContext(),
requireContext().applicationContext!!.packageName            +
".provider", file)
        } catch (e: java.lang.Exception) {
            e.message
        }
        return uri
    }

    fun takeScreenshot(mGL: GL10?): Bitmap? {
        val mWidth: Int = surfaceView!!.width
        val mHeight: Int = surfaceView!!.height
```

```
            val ib: IntBuffer = IntBuffer.allocate(mWidth *
mHeight)
            val ibt: IntBuffer = IntBuffer.allocate(mWidth *
mHeight)
        mGL!!.glReadPixels(0, 0, mWidth, mHeight, GL10.GL_RGBA,
GL10.GL_UNSIGNED_BYTE, ib)

            // Convert upside down mirror-reversed image to
right-side up normal image.
        for (i in 0 until mHeight) {
            for (j in 0 until mWidth) {
                    ibt.put((mHeight - i - 1) * mWidth + j,
ib.get(i * mWidth + j))
            }
        }
            val mBitmap = Bitmap.createBitmap(mWidth, mHeight,
Bitmap.Config.ARGB_8888)
        mBitmap.copyPixelsFromBuffer(ibt)
        return mBitmap
    }

      override fun onSurfaceChanged(gl: GL10?, width: Int,
height: Int) {
        displayRotationHelper.onSurfaceChanged(width, height);
        GLES20.glViewport(0, 0, width, height);
    }

        override fun onSurfaceCreated(gl: GL10?, config:
EGLConfig?) {
        GLES20.glClearColor(0.1f, 0.1f, 0.1f, 1.0f)
```

```
        // Prepare the rendering objects. This involves reading
shaders, so may throw an IOException.

        // Prepare the rendering objects. This involves reading
shaders, so may throw an IOException.
        try {
            // Create the texture and pass it to ARCore session
to be filled during update().
            backgroundRenderer.createOnGlThread(context)
        } catch (e: IOException) {
            println("Failed to read an asset file $e")
        }
    }

    private fun configureSession() {
        val config = Config(session)
                                    config.augmentedFaceMode     =
Config.AugmentedFaceMode.MESH3D
        session?.configure(config)
    }
}
```

```
package com.maquiAR.arface

public interface AugmentedFaceListener {
    fun onFaceAdded(face: AugmentedFaceNode)
    fun onFaceUpdate(face: AugmentedFaceNode)
}
```

```
package com.maquiAR.arface

import android.content.Context
```

```
import com.google.ar.core.AugmentedFace
import com.google.ar.core.Pose
import com.google.ar.core.TrackingState

class AugmentedFaceNode(private val augmentedFace:
AugmentedFace?, val context: Context) {

   private val augmentedFaceRenderer = AugmentedFaceRenderer()
       private val faceLandmarks = HashMap<FaceLandmark,
FaceRegion>()
   private var renderFaceMesh: Boolean = false

   companion object {
       enum class FaceLandmark {
           FOREHEAD_RIGHT,
           FOREHEAD_LEFT,
           NOSE_TIP
       }
   }

   init {
       renderFaceMesh = false
        augmentedFaceRenderer.setMaterialProperties(0.0f, 1.0f,
0.1f, 6.0f)
   }

    fun setRegionModel(faceLandmark: FaceLandmark, modelName:
String, modelTexture: String) {
       val faceRegion = FaceRegion(faceLandmark)
                faceRegion.setRenderable(context, modelName,
modelTexture)
       faceLandmarks[faceLandmark] = faceRegion
```

```
    }

    fun setFaceMeshTexture(assetNames: ArrayList<String>) {
                augmentedFaceRenderer.createOnGlThread(context,
assetNames)
        renderFaceMesh = true
    }

    fun setFaceMeshTexture(assetName: String) {
                augmentedFaceRenderer.createOnGlThread(context,
assetName)
        renderFaceMesh = true
    }

        fun  onDraw(projectionMatrix:  FloatArray,  viewMatrix:
FloatArray, colorCorrectionRgba: FloatArray) {
        augmentedFace?.let { face ->
            if (face.trackingState != TrackingState.TRACKING) {
                return
            }

            if (renderFaceMesh) {
                val modelMatrix = FloatArray(16)
                face.centerPose.toMatrix(modelMatrix, 0)
                augmentedFaceRenderer.draw(
                                    projectionMatrix,  viewMatrix,
modelMatrix, colorCorrectionRgba, face
                )
            }

            for (region in faceLandmarks.values) {
                val objectMatrix = FloatArray(16)
```

```
getRegionPose(region.faceLandmark)?.toMatrix(objectMatrix, 0)
                            region.draw(objectMatrix,  viewMatrix,
projectionMatrix, colorCorrectionRgba)
            }
        }
    }

    fun setContourColor(contourColor: FloatArray) {
        augmentedFaceRenderer.setContourColor(contourColor)
    }

    fun setEyeshadowColor(eyeshadowColor: FloatArray){
        augmentedFaceRenderer.setEyeshadowColor(eyeshadowColor)
    }

    fun setTextureColor(color: FloatArray) {
        augmentedFaceRenderer.setTextureColor(color)
    }

      private  fun  getRegionPose(faceLandmark:  FaceLandmark)  :
Pose? {
        return when (faceLandmark) {
                                        FaceLandmark.NOSE_TIP    ->
augmentedFace?.getRegionPose(AugmentedFace.RegionType.NOSE_TIP
)
                                    FaceLandmark.FOREHEAD_LEFT    ->
augmentedFace?.getRegionPose(AugmentedFace.RegionType.FOREHEAD
_LEFT)
                                   FaceLandmark.FOREHEAD_RIGHT    ->
augmentedFace?.getRegionPose(AugmentedFace.RegionType.FOREHEAD
_RIGHT)
```

```
        }
    }
}

package com.maquiAR.arface

import android.content.Context
import android.graphics.BitmapFactory
import android.opengl.GLES20
import android.opengl.GLUtils
import android.opengl.Matrix
import com.maquiAR.arface.rendering.ShaderUtil.loadGLShader
import com.google.ar.core.AugmentedFace
import java.io.IOException
import java.nio.FloatBuffer
import java.nio.ShortBuffer

public class AugmentedFaceRenderer {
                        private        val        TAG        =
AugmentedFaceRenderer::class.java.simpleName

    private var modelViewUniform = 0
    private var modelViewProjectionUniform = 0

    private var textureUniform = 0
        private var textureUniformEye = 0


    private var lightingParametersUniform = 0

    private var materialParametersUniform = 0
```

```
    private var colorCorrectionParameterUniform = 0

    private var tintColorUniform = 0
     private var tintColor = floatArrayOf(0.647f, 0.357f, 0.33f,
1f)

    private var tintColorEyesUniform = 0
     private var tintColorEyes = floatArrayOf(1.0f, 1.0f, 1.0f,
0f)

    private var attriVertices = 0
    private var attriUvs = 0
    private var attriNormals = 0

       // Set some default material properties to use for
lighting.
    private var ambient = 0.3f
    private var diffuse = 1.0f
    private var specular = 1.0f
    private var specularPower = 6.0f

    private val textureId = IntArray(1)

     private val lightDirection = floatArrayOf(0.0f, 1.0f, 0.0f,
0.0f)
    private var program = 0
    private val modelViewProjectionMat = FloatArray(16)
    private val modelViewMat = FloatArray(16)
    private val viewLightDirection = FloatArray(4)

    fun AugmentedFaceRenderer() {}
```

```
    @Throws(IOException::class)
    fun createOnGlThread(
        context: Context,
        diffuseTextureAssetNames: ArrayList<String>
    ) {
        val vertexShader: Int =
             loadGLShader(TAG, context, GLES20.GL_VERTEX_SHADER,
Companion.VERTEX_SHADER_NAME)
        val fragmentShader: Int =
                                    loadGLShader(TAG,   context,
GLES20.GL_FRAGMENT_SHADER, Companion.FRAGMENT_SHADER_NAME)
        program = GLES20.glCreateProgram()
        GLES20.glAttachShader(program, vertexShader)
        GLES20.glAttachShader(program, fragmentShader)
        GLES20.glLinkProgram(program)
                                  modelViewProjectionUniform    =
GLES20.glGetUniformLocation(program, "u_ModelViewProjection")
         modelViewUniform = GLES20.glGetUniformLocation(program,
"u_ModelView")
          textureUniform = GLES20.glGetUniformLocation(program,
"u_Texture")
        //Bind Eye texture with shader variable
                                       textureUniformEye     =
GLES20.glGetUniformLocation(program, "u_TextureEye")
                                   lightingParametersUniform    =
GLES20.glGetUniformLocation(program, "u_LightningParameters")
                                   materialParametersUniform    =
GLES20.glGetUniformLocation(program, "u_MaterialParameters")
        colorCorrectionParameterUniform =
                              GLES20.glGetUniformLocation(program,
"u_ColorCorrectionParameters")
```

```
        tintColorUniform = GLES20.glGetUniformLocation(program,
"u_TintColor")
                                    tintColorEyesUniform     =
GLES20.glGetUniformLocation(program, "u_TintColorEyes")
           attriVertices  =  GLES20.glGetAttribLocation(program,
"a_Position")
               attriUvs   =   GLES20.glGetAttribLocation(program,
"a_TexCoord")
            attriNormals  =  GLES20.glGetAttribLocation(program,
"a_Normal")
        // Size of the texture array
        GLES20.glGenTextures(2, textureId, 0)
        // Load texture0
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0)
                            loadTexture(context,    textureId,
diffuseTextureAssetNames[0], 0)

        // Load texture1
        GLES20.glActiveTexture(GLES20.GL_TEXTURE1)
                            loadTexture(context,    textureId,
diffuseTextureAssetNames[1], 1)
    }

    @Throws(IOException::class)
    private fun loadTexture(
        context: Context,
        textureId: IntArray,
        filename: String,
        index: Int
    ) {
                                    val     textureBitmap     =
BitmapFactory.decodeStream(context.getAssets().open(filename))
```

```
                    GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
textureId[index])
                    GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D,
GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR_MIPMAP_LINEAR)
                    GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D,
GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR)
                   GLUtils.texImage2D(GLES20.GL_TEXTURE_2D,   0,
textureBitmap, 0)
        GLES20.glGenerateMipmap(GLES20.GL_TEXTURE_2D)
                    GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
textureId[index])
        textureBitmap.recycle()
    }

    fun draw(
        projmtx: FloatArray?,
        viewmtx: FloatArray?,
        modelmtx: FloatArray?,
        colorCorrectionRgba: FloatArray?,
        face: AugmentedFace
    ) {
        val vertices: FloatBuffer = face.meshVertices
        val normals: FloatBuffer = face.meshNormals
                        val   textureCoords:   FloatBuffer   =
face.meshTextureCoordinates
                    val   triangleIndices:   ShortBuffer   =
face.meshTriangleIndices
        GLES20.glUseProgram(program)
        GLES20.glDepthMask(false)
        val modelViewProjectionMatTemp = FloatArray(16)
              Matrix.multiplyMM(modelViewProjectionMatTemp,  0,
projmtx, 0, viewmtx, 0)
```

```
                Matrix.multiplyMM(modelViewProjectionMat,  0,
modelViewProjectionMatTemp, 0, modelmtx, 0)
             Matrix.multiplyMM(modelViewMat,  0,  viewmtx,  0,
modelmtx, 0)

        // Set the lighting environment properties.
         Matrix.multiplyMV(viewLightDirection, 0, modelViewMat,
0, lightDirection, 0)
        normalizeVec3(viewLightDirection)
        GLES20.glUniform4f(
            lightingParametersUniform,
            viewLightDirection[0],
            viewLightDirection[1],
            viewLightDirection[2],
            1f
        )
         GLES20.glUniform4fv(colorCorrectionParameterUniform, 1,
colorCorrectionRgba, 0)

        // Set the object material properties.
          GLES20.glUniform4f(materialParametersUniform, ambient,
diffuse, specular, specularPower)

        // Set the ModelViewProjection matrix in the shader.
          GLES20.glUniformMatrix4fv(modelViewUniform, 1, false,
modelViewMat, 0)
           GLES20.glUniformMatrix4fv(modelViewProjectionUniform,
1, false, modelViewProjectionMat, 0)
        GLES20.glEnableVertexAttribArray(attriVertices)
                GLES20.glVertexAttribPointer(attriVertices,  3,
GLES20.GL_FLOAT, false, 0, vertices)
        GLES20.glEnableVertexAttribArray(attriNormals)
```

```
                GLES20.glVertexAttribPointer(attriNormals,  3,
GLES20.GL_FLOAT, false, 0, normals)
        GLES20.glEnableVertexAttribArray(attriUvs)
                    GLES20.glVertexAttribPointer(attriUvs,  2,
GLES20.GL_FLOAT, false, 0, textureCoords)
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0)
        GLES20.glUniform1i(textureUniform, 0)
                     GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
textureId[0])
        GLES20.glUniform4fv(tintColorUniform, 1, tintColor, 0)
        GLES20.glEnable(GLES20.GL_BLEND)

        // Textures are loaded with premultiplied alpha
                                                                //
(https://developer.android.com/reference/android/graphics/Bitm
apFactory.Options#inPremultiplied),
        // so we use the premultiplied alpha blend factors.
                                GLES20.glBlendFunc(GLES20.GL_ONE,
GLES20.GL_ONE_MINUS_SRC_ALPHA)
        GLES20.glDrawElements(
                  GLES20.GL_TRIANGLES,  triangleIndices.limit(),
GLES20.GL_UNSIGNED_SHORT, triangleIndices
        )
        GLES20.glUseProgram(0)
        GLES20.glDepthMask(true)
    }

    @Throws(IOException::class)
    fun createOnGlThread(
        context: Context,
        diffuseTextureAssetName: String
    ) {
```

```
        val vertexShader: Int =
                loadGLShader(TAG, context, GLES20.GL_VERTEX_SHADER,
Companion.VERTEX_SHADER_NAME)
        val fragmentShader: Int =
                                        loadGLShader(TAG,   context,
GLES20.GL_FRAGMENT_SHADER, Companion.FRAGMENT_SHADER_NAME)
        program = GLES20.glCreateProgram()
        GLES20.glAttachShader(program, vertexShader)
        GLES20.glAttachShader(program, fragmentShader)
        GLES20.glLinkProgram(program)
                                    modelViewProjectionUniform    =
GLES20.glGetUniformLocation(program, "u_ModelViewProjection")
         modelViewUniform = GLES20.glGetUniformLocation(program,
"u_ModelView")
           textureUniform  =  GLES20.glGetUniformLocation(program,
"u_Texture")
                                     lightingParametersUniform    =
GLES20.glGetUniformLocation(program, "u_LightningParameters")
                                    materialParametersUniform    =
GLES20.glGetUniformLocation(program, "u_MaterialParameters")
        colorCorrectionParameterUniform =
                               GLES20.glGetUniformLocation(program,
"u_ColorCorrectionParameters")
         tintColorUniform = GLES20.glGetUniformLocation(program,
"u_TintColor")
            attriVertices  =  GLES20.glGetAttribLocation(program,
"a_Position")
                attriUvs  =  GLES20.glGetAttribLocation(program,
"a_TexCoord")
             attriNormals  =  GLES20.glGetAttribLocation(program,
"a_Normal")
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0)
```

```
        GLES20.glGenTextures(1, textureId, 0)
                                loadTexture(context,    textureId,
diffuseTextureAssetName)
    }

    @Throws(IOException::class)
    private fun loadTexture(
        context: Context,
        textureId: IntArray,
        filename: String
    ) {
                                        val     textureBitmap     =
BitmapFactory.decodeStream(context.getAssets().open(filename))
                        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
textureId[0])
        GLES20.glTexParameteri(
             GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER,
GLES20.GL_LINEAR_MIPMAP_LINEAR
        )
                     GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D,
GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR)
                    GLUtils.texImage2D(GLES20.GL_TEXTURE_2D,   0,
textureBitmap, 0)
        GLES20.glGenerateMipmap(GLES20.GL_TEXTURE_2D)
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, 0)
        textureBitmap.recycle()
    }


    fun setMaterialProperties(
        ambient: Float,
        diffuse: Float,
```

```
        specular: Float,
        specularPower: Float
    ) {
        this.ambient = ambient
        this.diffuse = diffuse
        this.specular = specular
        this.specularPower = specularPower
    }

    private fun normalizeVec3(v: FloatArray) {
        val reciprocalLength = 1.0f / Math.sqrt(
                    v[0] * v[0] + v[1] * v[1] + (v[2] *
v[2]).toDouble()
        ).toFloat()
        v[0] *= reciprocalLength
        v[1] *= reciprocalLength
        v[2] *= reciprocalLength
    }

    fun setContourColor(contourColor: FloatArray) {
        tintColor = contourColor
    }

    fun setEyeshadowColor(eyeshadowColor: FloatArray) {
        tintColorEyes = eyeshadowColor
    }

    fun setTextureColor(color: FloatArray) {
        tintColor = color
    }

    companion object {
```

```
                private const val VERTEX_SHADER_NAME =
"shaders/object.vert"
                private const val FRAGMENT_SHADER_NAME =
"shaders/object.frag"
    }
}

package com.maquiAR.arface

import android.content.Context
import com.maquiAR.arface.rendering.ObjectRenderer

class FaceRegion(val faceLandmark:
AugmentedFaceNode.Companion.FaceLandmark) {
        private val objectRenderer: ObjectRenderer =
ObjectRenderer()
    private val scaleFactor = 1.0f

    fun setRenderable (context: Context, modelName: String,
modelTexture: String) {
        objectRenderer.createOnGlThread( /*context=*/context,
            modelName,
            modelTexture
        )
          objectRenderer.setMaterialProperties(0.0f, 1.0f, 0.1f,
6.0f)

objectRenderer.setBlendMode(ObjectRenderer.BlendMode.AlphaBlen
ding)
    }
```

```
    fun draw(objectMatrix: FloatArray, viewMatrix: FloatArray,
projectionMatrix: FloatArray, colorCorrectionRgba: FloatArray)
{
                objectRenderer.updateModelMatrix(objectMatrix,
scaleFactor)
      objectRenderer.draw(
          viewMatrix,
          projectionMatrix,
          colorCorrectionRgba,
          DEFAULT_COLOR
      )
   }

   companion object {
         private val DEFAULT_COLOR = floatArrayOf(0f, 0f, 0f,
0f)
   }
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed under the Apache License, Version 2.0 (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
```

```
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
* See the License for the specific language governing
permissions and
* limitations under the License.
*/
package com.maquiAR.arface.rendering;

import android.content.Context;
import android.opengl.GLES11Ext;
import android.opengl.GLES20;

import androidx.annotation.NonNull;

import com.google.ar.core.Coordinates2d;
import com.google.ar.core.Frame;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;

/**
* This class renders the AR background from camera feed. It
creates and hosts the texture given to
* ARCore to be filled with the camera image.
*/
public class BackgroundRenderer {
            private static final String TAG =
BackgroundRenderer.class.getSimpleName();
```

```
    // Shader names.
     private static final String CAMERA_VERTEX_SHADER_NAME =
"shaders/screenquad.vert";
    private static final String CAMERA_FRAGMENT_SHADER_NAME =
"shaders/screenquad.frag";
                    private      static      final      String
DEPTH_VISUALIZER_VERTEX_SHADER_NAME =

"shaders/background_show_depth_color_visualization.vert";
                    private      static      final      String
DEPTH_VISUALIZER_FRAGMENT_SHADER_NAME =

"shaders/background_show_depth_color_visualization.frag";

    private static final int COORDS_PER_VERTEX = 2;
    private static final int TEXCOORDS_PER_VERTEX = 2;
    private static final int FLOAT_SIZE = 4;

    private FloatBuffer quadCoords;
    private FloatBuffer quadTexCoords;

    private int cameraProgram;
    private int depthProgram;

    private int cameraPositionAttrib;
    private int cameraTexCoordAttrib;
    private int cameraTextureUniform;
    private int cameraTextureId = -1;
    private boolean suppressTimestampZeroRendering = true;

    private int depthPositionAttrib;
```

```
    private int depthTexCoordAttrib;
    private int depthTextureUniform;
    private int depthTextureId = -1;

    public int getTextureId() {
        return cameraTextureId;
    }

    /**
     * Allocates and initializes OpenGL resources needed by the
background renderer. Must be called on
             *   the   OpenGL   thread,   typically   in   {@link
GLSurfaceView.Renderer#onSurfaceCreated(GL10,
     * EGLConfig)}.
     *
     * @param context Needed to access shader source.
     */
         public   void   createOnGlThread(Context   context,   int
depthTextureId) throws IOException {
        // Generate the background texture.
        int[] textures = new int[1];
        GLES20.glGenTextures(1, textures, 0);
        cameraTextureId = textures[0];
        int textureTarget = GLES11Ext.GL_TEXTURE_EXTERNAL_OES;
        GLES20.glBindTexture(textureTarget, cameraTextureId);
                            GLES20.glTexParameteri(textureTarget,
GLES20.GL_TEXTURE_WRAP_S, GLES20.GL_CLAMP_TO_EDGE);
                            GLES20.glTexParameteri(textureTarget,
GLES20.GL_TEXTURE_WRAP_T, GLES20.GL_CLAMP_TO_EDGE);
                            GLES20.glTexParameteri(textureTarget,
GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR);
```

```
                        GLES20.glTexParameteri(textureTarget,
GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR);

        int numVertices = 4;
                if (numVertices != QUAD_COORDS.length /
COORDS_PER_VERTEX) {
            throw new RuntimeException("Unexpected number of
vertices in BackgroundRenderer.");
        }

                                ByteBuffer bbCoords =
ByteBuffer.allocateDirect(QUAD_COORDS.length * FLOAT_SIZE);
        bbCoords.order(ByteOrder.nativeOrder());
        quadCoords = bbCoords.asFloatBuffer();
        quadCoords.put(QUAD_COORDS);
        quadCoords.position(0);

        ByteBuffer bbTexCoordsTransformed =
                    ByteBuffer.allocateDirect(numVertices *
TEXCOORDS_PER_VERTEX * FLOAT_SIZE);
        bbTexCoordsTransformed.order(ByteOrder.nativeOrder());
        quadTexCoords = bbTexCoordsTransformed.asFloatBuffer();

        // Load render camera feed shader.
        {
            int vertexShader =
                    ShaderUtil.loadGLShader(TAG, context,
GLES20.GL_VERTEX_SHADER, CAMERA_VERTEX_SHADER_NAME);
            int fragmentShader =
                    ShaderUtil.loadGLShader(
                                                TAG, context,
GLES20.GL_FRAGMENT_SHADER, CAMERA_FRAGMENT_SHADER_NAME);
```

```
            cameraProgram = GLES20.glCreateProgram();
            GLES20.glAttachShader(cameraProgram, vertexShader);
                                GLES20.glAttachShader(cameraProgram,
fragmentShader);
            GLES20.glLinkProgram(cameraProgram);
            GLES20.glUseProgram(cameraProgram);
                                            cameraPositionAttrib    =
GLES20.glGetAttribLocation(cameraProgram, "a_Position");
                                            cameraTexCoordAttrib    =
GLES20.glGetAttribLocation(cameraProgram, "a_TexCoord");
            ShaderUtil.checkGLError(TAG, "Program creation");

                                            cameraTextureUniform    =
GLES20.glGetUniformLocation(cameraProgram, "sTexture");
            ShaderUtil.checkGLError(TAG, "Program parameters");
        }

        // Load render depth map shader.
        {
            int vertexShader =
                    ShaderUtil.loadGLShader(
                                                    TAG,  context,
GLES20.GL_VERTEX_SHADER, DEPTH_VISUALIZER_VERTEX_SHADER_NAME);
            int fragmentShader =
                    ShaderUtil.loadGLShader(
                                                    TAG,  context,
GLES20.GL_FRAGMENT_SHADER,
DEPTH_VISUALIZER_FRAGMENT_SHADER_NAME);

            depthProgram = GLES20.glCreateProgram();
            GLES20.glAttachShader(depthProgram, vertexShader);
```

```
                                GLES20.glAttachShader(depthProgram,
fragmentShader);
            GLES20.glLinkProgram(depthProgram);
            GLES20.glUseProgram(depthProgram);
                                            depthPositionAttrib    =
GLES20.glGetAttribLocation(depthProgram, "a_Position");
                                            depthTexCoordAttrib    =
GLES20.glGetAttribLocation(depthProgram, "a_TexCoord");
            ShaderUtil.checkGLError(TAG, "Program creation");

                                            depthTextureUniform    =
GLES20.glGetUniformLocation(depthProgram, "u_DepthTexture");
            ShaderUtil.checkGLError(TAG, "Program parameters");
        }

        this.depthTextureId = depthTextureId;
    }

       public  void  createOnGlThread(Context  context)  throws
IOException {
        createOnGlThread(context, /*depthTextureId=*/ -1);
    }

          public   void   suppressTimestampZeroRendering(boolean
suppressTimestampZeroRendering) {
                        this.suppressTimestampZeroRendering    =
suppressTimestampZeroRendering;
    }

    /**
     * Draws the AR background image. The image will be drawn
such that virtual content rendered with
```

```
                * the matrices provided by {@link
com.google.ar.core.Camera#getViewMatrix(float[], int)} and
                                                * {@link
com.google.ar.core.Camera#getProjectionMatrix(float[], int,
float, float)} will
     * accurately follow static physical objects. This must be
called <b>before</b> drawing virtual
    * content.
    *
     * @param frame The current {@code Frame} as returned by
{@link Session#update()}.
      * @param debugShowDepthMap Toggles whether to show the
live camera feed or latest depth image.
    */
        public void draw(@NonNull Frame frame, boolean
debugShowDepthMap) {
       // If display rotation changed (also includes view size
change), we need to re-query the uv
        // coordinates for the screen rect, as they may have
changed as well.
       if (frame.hasDisplayGeometryChanged()) {
           frame.transformCoordinates2d(

Coordinates2d.OPENGL_NORMALIZED_DEVICE_COORDINATES,
                   quadCoords,
                   Coordinates2d.TEXTURE_NORMALIZED,
                   quadTexCoords);
       }

                    if (frame.getTimestamp() == 0 &&
suppressTimestampZeroRendering) {
```

```
            // Suppress rendering if the camera did not produce
the first frame yet. This is to avoid
                // drawing possible leftover data from previous
sessions if the texture is reused.
            return;
        }

        draw(debugShowDepthMap);
    }

    public void draw(@NonNull Frame frame) {
        draw(frame, /*debugShowDepthMap=*/ false);
    }

    /**
     * Draws the camera image using the currently configured
{@link BackgroundRenderer#quadTexCoords}
     * image texture coordinates.
     *
     * <p>The image will be center cropped if the camera sensor
aspect ratio does not match the screen
     * aspect ratio, which matches the cropping behavior of
{@link
     * Frame#transformCoordinates2d(Coordinates2d, float[],
Coordinates2d, float[])}.
     */
    public void draw(
                    int imageWidth, int imageHeight, float
screenAspectRatio, int cameraToDisplayRotation) {
        // Crop the camera image to fit the screen aspect
ratio.
```

```
            float imageAspectRatio = (float) imageWidth /
imageHeight;
        float croppedWidth;
        float croppedHeight;
        if (screenAspectRatio < imageAspectRatio) {
            croppedWidth = imageHeight * screenAspectRatio;
            croppedHeight = imageHeight;
        } else {
            croppedWidth = imageWidth;
            croppedHeight = imageWidth / screenAspectRatio;
        }

          float u = (imageWidth - croppedWidth) / imageWidth *
0.5f;
         float v = (imageHeight - croppedHeight) / imageHeight *
0.5f;

        float[] texCoordTransformed;
        switch (cameraToDisplayRotation) {
            case 90:
                  texCoordTransformed = new float[] {1 - u, 1 -
v, 1 - u, v, u, 1 - v, u, v};
                break;
            case 180:
                 texCoordTransformed = new float[] {1 - u, v, u,
v, 1 - u, 1 - v, u, 1 - v};
                break;
            case 270:
                 texCoordTransformed = new float[] {u, v, u, 1 -
v, 1 - u, v, 1 - u, 1 - v};
                break;
            case 0:
```

```
                texCoordTransformed = new float[] {u, 1 - v, 1
- u, 1 - v, u, v, 1 - u, v};
                break;
            default:
                throw new IllegalArgumentException("Unhandled
rotation: " + cameraToDisplayRotation);
        }

        // Write image texture coordinates.
        quadTexCoords.position(0);
        quadTexCoords.put(texCoordTransformed);

        draw(/*debugShowDepthMap=*/ false);
    }

    /**
     * Draws the camera background image using the currently
configured {@link
     *  BackgroundRenderer#quadTexCoords}  image  texture
coordinates.
     */
    private void draw(boolean debugShowDepthMap) {
        // Ensure position is rewound before use.
        quadTexCoords.position(0);

        // No need to test or write depth, the screen quad has
arbitrary depth, and is expected
        // to be drawn first.
        GLES20.glDisable(GLES20.GL_DEPTH_TEST);
        GLES20.glDepthMask(false);

        GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
```

```
        if (debugShowDepthMap) {
                        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
depthTextureId);
            GLES20.glUseProgram(depthProgram);
            GLES20.glUniform1i(depthTextureUniform, 0);

                    // Set the vertex positions and texture
coordinates.
            GLES20.glVertexAttribPointer(
                        depthPositionAttrib, COORDS_PER_VERTEX,
GLES20.GL_FLOAT, false, 0, quadCoords);
            GLES20.glVertexAttribPointer(
                    depthTexCoordAttrib, TEXCOORDS_PER_VERTEX,
GLES20.GL_FLOAT, false, 0, quadTexCoords);

GLES20.glEnableVertexAttribArray(depthPositionAttrib);

GLES20.glEnableVertexAttribArray(depthTexCoordAttrib);
        } else {

GLES20.glBindTexture(GLES11Ext.GL_TEXTURE_EXTERNAL_OES,
cameraTextureId);
            GLES20.glUseProgram(cameraProgram);
            GLES20.glUniform1i(cameraTextureUniform, 0);

                    // Set the vertex positions and texture
coordinates.
            GLES20.glVertexAttribPointer(
                    cameraPositionAttrib, COORDS_PER_VERTEX,
GLES20.GL_FLOAT, false, 0, quadCoords);
            GLES20.glVertexAttribPointer(
```

```
                    cameraTexCoordAttrib, TEXCOORDS_PER_VERTEX,
GLES20.GL_FLOAT, false, 0, quadTexCoords);

GLES20.glEnableVertexAttribArray(cameraPositionAttrib);

GLES20.glEnableVertexAttribArray(cameraTexCoordAttrib);
        }

        GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4);

        // Disable vertex arrays
        if (debugShowDepthMap) {

GLES20.glDisableVertexAttribArray(depthPositionAttrib);

GLES20.glDisableVertexAttribArray(depthTexCoordAttrib);
        } else {

GLES20.glDisableVertexAttribArray(cameraPositionAttrib);

GLES20.glDisableVertexAttribArray(cameraTexCoordAttrib);
        }

        // Restore the depth state for further drawing.
        GLES20.glDepthMask(true);
        GLES20.glEnable(GLES20.GL_DEPTH_TEST);

        ShaderUtil.checkGLError(TAG, "BackgroundRendererDraw");
    }

    /**
     * (-1, 1) ------- (1, 1)
```

```
     *   |     \            |
     *   |        \         |
     *   |           \      |
     *   |              \   |
     * (-1, -1) ------ (1, -1)
          * Ensure triangles are front-facing, to support
glCullFace().
      * This quad will be drawn using GL_TRIANGLE_STRIP which
draws two
     * triangles: v0->v1->v2, then v2->v1->v3.
     */
    private static final float[] QUAD_COORDS =
            new float[] {
                    -1.0f, -1.0f, +1.0f, -1.0f, -1.0f, +1.0f,
+1.0f, +1.0f,
            };
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed under the Apache License, Version 2.0 (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
```

```
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
* See the License for the specific language governing
permissions and
* limitations under the License.
*/
package com.maquiAR.arface.rendering;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLES20;
import android.opengl.GLUtils;
import android.opengl.Matrix;

import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.IntBuffer;
import java.nio.ShortBuffer;
import java.util.Map;
import java.util.TreeMap;

import de.javagl.obj.Obj;
import de.javagl.obj.ObjData;
import de.javagl.obj.ObjReader;
import de.javagl.obj.ObjUtils;

/** Renders an object loaded from an OBJ file in OpenGL. */
public class ObjectRenderer {
```

```
            private    static    final    String    TAG    =
ObjectRenderer.class.getSimpleName();

    /**
     * Blend mode.
     *
     * @see #setBlendMode(BlendMode)
     */
    public enum BlendMode {
            /** Multiplies the destination color by the source
alpha, without z-buffer writing. */
        Shadow,
        /** Normal alpha blending with z-buffer writing. */
        AlphaBlending
    }

    // Shader names.
         private  static  final  String  VERTEX_SHADER_NAME  =
"shaders/ar_object.vert";
       private  static  final  String  FRAGMENT_SHADER_NAME  =
"shaders/ar_object.frag";

    private static final int COORDS_PER_VERTEX = 3;
     private  static  final  float[]  DEFAULT_COLOR  =  new  float[]
{0f, 0f, 0f, 0f};

    // Note: the last component must be zero to avoid applying
the translational part of the matrix.
    private static final float[] LIGHT_DIRECTION = new float[]
{0.250f, 0.866f, 0.433f, 0.0f};
    private final float[] viewLightDirection = new float[4];
```

```
// Object vertex buffer variables.
private int vertexBufferId;
private int verticesBaseAddress;
private int texCoordsBaseAddress;
private int normalsBaseAddress;
private int indexBufferId;
private int indexCount;

private int program;
private final int[] textures = new int[1];

// Shader location: model view projection matrix.
private int modelViewUniform;
private int modelViewProjectionUniform;

// Shader location: object attributes.
private int positionAttribute;
private int normalAttribute;
private int texCoordAttribute;

// Shader location: texture sampler.
private int textureUniform;

// Shader location: environment properties.
private int lightingParametersUniform;

// Shader location: material properties.
private int materialParametersUniform;

// Shader location: color correction property.
private int colorCorrectionParameterUniform;
```

```
    // Shader location: object color property (to change the
primary color of the object).
    private int colorUniform;

    // Shader location: depth texture.
    private int depthTextureUniform;

    // Shader location: transform to depth uvs.
    private int depthUvTransformUniform;

    // Shader location: the aspect ratio of the depth texture.
    private int depthAspectRatioUniform;

    private BlendMode blendMode = null;

    // Temporary matrices allocated here to reduce number of
allocations for each frame.
    private final float[] modelMatrix = new float[16];
    private final float[] modelViewMatrix = new float[16];
      private final float[] modelViewProjectionMatrix = new
float[16];

      // Set some default material properties to use for
lighting.
    private float ambient = 0.3f;
    private float diffuse = 1.0f;
    private float specular = 1.0f;
    private float specularPower = 6.0f;

    // Depth-for-Occlusion parameters.
```

```
                private static final String
USE_DEPTH_FOR_OCCLUSION_SHADER_FLAG =
"USE_DEPTH_FOR_OCCLUSION";
   private boolean useDepthForOcclusion = false;
   private float depthAspectRatio = 0.0f;
   private float[] uvTransform = null;
   private int depthTextureId;

   /**
      * Creates and initializes OpenGL resources needed for
rendering the model.
    *
      * @param context Context for loading the shader and
below-named model and texture assets.
     * @param objAssetName Name of the OBJ file containing the
model geometry.
      * @param diffuseTextureAssetName Name of the PNG file
containing the diffuse texture map.
    */
      public void createOnGlThread(Context context, String
objAssetName, String diffuseTextureAssetName)
          throws IOException {
        // Compiles and loads the shader based on the current
configuration.
       compileAndLoadShaderProgram(context);

       // Read the texture.
       Bitmap textureBitmap =

BitmapFactory.decodeStream(context.getAssets().open(diffuseTex
tureAssetName));
```

```
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
        GLES20.glGenTextures(textures.length, textures, 0);
                        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
textures[0]);

        GLES20.glTexParameteri(
                                                GLES20.GL_TEXTURE_2D,
GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR_MIPMAP_LINEAR);
                         GLES20.glTexParameteri(GLES20.GL_TEXTURE_2D,
GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR);
                        GLUtils.texImage2D(GLES20.GL_TEXTURE_2D,   0,
textureBitmap, 0);
        GLES20.glGenerateMipmap(GLES20.GL_TEXTURE_2D);
        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, 0);

        textureBitmap.recycle();

        ShaderUtil.checkGLError(TAG, "Texture loading");

        // Read the obj file.
                                InputStream    objInputStream    =
context.getAssets().open(objAssetName);
        Obj obj = ObjReader.read(objInputStream);

          // Prepare the Obj so that its structure is suitable
for
        // rendering with OpenGL:
        // 1. Triangulate it
            // 2. Make sure that texture coordinates are not
ambiguous
        // 3. Make sure that normals are not ambiguous
        // 4. Convert it to single-indexed data
```

```
        obj = ObjUtils.convertToRenderable(obj);

          // OpenGL does not use Java arrays. ByteBuffers are
used instead to provide data in a format
        // that OpenGL understands.

        // Obtain the data from the OBJ, as direct buffers:
                                    IntBuffer    wideIndices    =
ObjData.getFaceVertexIndices(obj, 3);
        FloatBuffer vertices = ObjData.getVertices(obj);
        FloatBuffer texCoords = ObjData.getTexCoords(obj, 2);
        FloatBuffer normals = ObjData.getNormals(obj);

            // Convert int indices to shorts for GL ES 2.0
compatibility
        ShortBuffer indices =
                                    ByteBuffer.allocateDirect(2  *
wideIndices.limit())
                        .order(ByteOrder.nativeOrder())
                        .asShortBuffer();
        while (wideIndices.hasRemaining()) {
            indices.put((short) wideIndices.get());
        }
        indices.rewind();

        int[] buffers = new int[2];
        GLES20.glGenBuffers(2, buffers, 0);
        vertexBufferId = buffers[0];
        indexBufferId = buffers[1];

        // Load vertex buffer
        verticesBaseAddress = 0;
```

```
            texCoordsBaseAddress = verticesBaseAddress + 4 *
vertices.limit();
             normalsBaseAddress = texCoordsBaseAddress + 4 *
texCoords.limit();
            final int totalBytes = normalsBaseAddress + 4 *
normals.limit();

                    GLES20.glBindBuffer(GLES20.GL_ARRAY_BUFFER,
vertexBufferId);
        GLES20.glBufferData(GLES20.GL_ARRAY_BUFFER, totalBytes,
null, GLES20.GL_STATIC_DRAW);
       GLES20.glBufferSubData(
                GLES20.GL_ARRAY_BUFFER, verticesBaseAddress, 4
* vertices.limit(), vertices);
       GLES20.glBufferSubData(
                GLES20.GL_ARRAY_BUFFER, texCoordsBaseAddress, 4
* texCoords.limit(), texCoords);
       GLES20.glBufferSubData(
                GLES20.GL_ARRAY_BUFFER, normalsBaseAddress, 4 *
normals.limit(), normals);
       GLES20.glBindBuffer(GLES20.GL_ARRAY_BUFFER, 0);

       // Load index buffer
            GLES20.glBindBuffer(GLES20.GL_ELEMENT_ARRAY_BUFFER,
indexBufferId);
       indexCount = indices.limit();
       GLES20.glBufferData(
                GLES20.GL_ELEMENT_ARRAY_BUFFER, 2 * indexCount,
indices, GLES20.GL_STATIC_DRAW);
       GLES20.glBindBuffer(GLES20.GL_ELEMENT_ARRAY_BUFFER, 0);

       ShaderUtil.checkGLError(TAG, "OBJ buffer load");
```

```
        Matrix.setIdentityM(modelMatrix, 0);
    }

    /**
     * Selects the blending mode for rendering.
     *
      * @param blendMode The blending mode. Null indicates no
blending (opaque rendering).
     */
    public void setBlendMode(BlendMode blendMode) {
        this.blendMode = blendMode;
    }

    /**
      * Specifies whether to use the depth texture to perform
depth-based occlusion of virtual objects
     * from real-world geometry.
     *
     * <p>This function is a no-op if the value provided is the
same as what is already set. If the
      * value changes, this function will recompile and reload
the shader program to either
           *  enable/disable  depth-based  occlusion.  NOTE:
recompilation of the shader is inefficient. This
      * code could be optimized to precompile both versions of
the shader.
     *
     * @param context Context for loading the shader.
     * @param useDepthForOcclusion Specifies whether to use the
depth texture to perform occlusion
     *     during rendering of virtual objects.
```

```
    */
       public  void  setUseDepthForOcclusion(Context  context,
boolean useDepthForOcclusion)
           throws IOException {
        if (this.useDepthForOcclusion == useDepthForOcclusion)
{
           return; // No change, does nothing.
       }

        // Toggles the occlusion rendering mode and recompiles
the shader.
       this.useDepthForOcclusion = useDepthForOcclusion;
       compileAndLoadShaderProgram(context);
   }

    private  void  compileAndLoadShaderProgram(Context  context)
throws IOException {
        // Compiles and loads the shader program based on the
selected mode.
       Map<String, Integer> defineValuesMap = new TreeMap<>();

defineValuesMap.put(USE_DEPTH_FOR_OCCLUSION_SHADER_FLAG,
useDepthForOcclusion ? 1 : 0);

       final int vertexShader =
                           ShaderUtil.loadGLShader(TAG,  context,
GLES20.GL_VERTEX_SHADER, VERTEX_SHADER_NAME);
       final int fragmentShader =
               ShaderUtil.loadGLShader(
                                                   TAG,  context,
GLES20.GL_FRAGMENT_SHADER,                FRAGMENT_SHADER_NAME,
defineValuesMap);
```

```
        program = GLES20.glCreateProgram();
        GLES20.glAttachShader(program, vertexShader);
        GLES20.glAttachShader(program, fragmentShader);
        GLES20.glLinkProgram(program);
        GLES20.glUseProgram(program);

        ShaderUtil.checkGLError(TAG, "Program creation");

         modelViewUniform = GLES20.glGetUniformLocation(program,
"u_ModelView");
                                  modelViewProjectionUniform    =
GLES20.glGetUniformLocation(program, "u_ModelViewProjection");

         positionAttribute = GLES20.glGetAttribLocation(program,
"a_Position");
          normalAttribute = GLES20.glGetAttribLocation(program,
"a_Normal");
         texCoordAttribute = GLES20.glGetAttribLocation(program,
"a_TexCoord");

          textureUniform = GLES20.glGetUniformLocation(program,
"u_Texture");

                                   lightingParametersUniform    =
GLES20.glGetUniformLocation(program, "u_LightingParameters");
                                   materialParametersUniform    =
GLES20.glGetUniformLocation(program, "u_MaterialParameters");
        colorCorrectionParameterUniform =
                              GLES20.glGetUniformLocation(program,
"u_ColorCorrectionParameters");
```

```
            colorUniform = GLES20.glGetUniformLocation(program,
"u_ObjColor");

        // Occlusion Uniforms.
        if (useDepthForOcclusion) {
                                        depthTextureUniform   =
GLES20.glGetUniformLocation(program, "u_DepthTexture");
                                    depthUvTransformUniform   =
GLES20.glGetUniformLocation(program, "u_DepthUvTransform");
                                    depthAspectRatioUniform   =
GLES20.glGetUniformLocation(program, "u_DepthAspectRatio");
        }

        ShaderUtil.checkGLError(TAG, "Program parameters");
    }

    /**
     * Updates the object model matrix and applies scaling.
     *
      * @param modelMatrix A 4x4 model-to-world transformation
matrix, stored in column-major order.
      * @param scaleFactor A separate scaling factor to apply
before the {@code modelMatrix}.
     * @see android.opengl.Matrix
     */
     public void updateModelMatrix(float[] modelMatrix, float
scaleFactor) {
        float[] scaleMatrix = new float[16];
        Matrix.setIdentityM(scaleMatrix, 0);
        scaleMatrix[0] = scaleFactor;
        scaleMatrix[5] = scaleFactor;
        scaleMatrix[10] = scaleFactor;
```

```
        Matrix.multiplyMM(this.modelMatrix, 0, modelMatrix, 0,
scaleMatrix, 0);
    }

    /**
     * Sets the surface characteristics of the rendered model.
     *
       * @param ambient Intensity of non-directional surface
illumination.
     * @param diffuse Diffuse (matte) surface reflectivity.
     * @param specular Specular (shiny) surface reflectivity.
      * @param specularPower Surface shininess. Larger values
result in a smaller, sharper specular
     *     highlight.
     */
    public void setMaterialProperties(
              float ambient, float diffuse, float specular, float
specularPower) {
        this.ambient = ambient;
        this.diffuse = diffuse;
        this.specular = specular;
        this.specularPower = specularPower;
    }

    /**
     * Draws the model.
     *
      * @param cameraView A 4x4 view matrix, in column-major
order.
      * @param cameraPerspective A 4x4 projection matrix, in
column-major order.
```

```
     * @param colorCorrectionRgba Illumination intensity.
Combined with diffuse and specular material
     *     properties.
     * @see #setBlendMode(BlendMode)
     * @see #updateModelMatrix(float[], float)
     * @see #setMaterialProperties(float, float, float, float)
     * @see android.opengl.Matrix
     */
          public    void    draw(float[]    cameraView,    float[]
cameraPerspective, float[] colorCorrectionRgba) {
                          draw(cameraView,    cameraPerspective,
colorCorrectionRgba, DEFAULT_COLOR);
   }

   public void draw(
           float[] cameraView,
           float[] cameraPerspective,
           float[] colorCorrectionRgba,
           float[] objColor) {

       ShaderUtil.checkGLError(TAG, "Before draw");

       // Build the ModelView and ModelViewProjection matrices
       // for calculating object position and light.
          Matrix.multiplyMM(modelViewMatrix, 0, cameraView, 0,
modelMatrix, 0);
               Matrix.multiplyMM(modelViewProjectionMatrix,  0,
cameraPerspective, 0, modelViewMatrix, 0);

       GLES20.glUseProgram(program);

       // Set the lighting environment properties.
```

```
                    Matrix.multiplyMV(viewLightDirection,   0,
modelViewMatrix, 0, LIGHT_DIRECTION, 0);
        normalizeVec3(viewLightDirection);
        GLES20.glUniform4f(
                lightingParametersUniform,
                viewLightDirection[0],
                viewLightDirection[1],
                viewLightDirection[2],
                1.f);
         GLES20.glUniform4fv(colorCorrectionParameterUniform, 1,
colorCorrectionRgba, 0);

        // Set the object color property.
        GLES20.glUniform4fv(colorUniform, 1, objColor, 0);

        // Set the object material properties.
          GLES20.glUniform4f(materialParametersUniform, ambient,
diffuse, specular, specularPower);

        // Attach the object texture.
        GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
                    GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
textures[0]);
        GLES20.glUniform1i(textureUniform, 0);

        // Occlusion parameters.
        if (useDepthForOcclusion) {
            // Attach the depth texture.
            GLES20.glActiveTexture(GLES20.GL_TEXTURE1);
                    GLES20.glBindTexture(GLES20.GL_TEXTURE_2D,
depthTextureId);
            GLES20.glUniform1i(depthTextureUniform, 1);
```

```
            // Set the depth texture uv transform.
              GLES20.glUniformMatrix3fv(depthUvTransformUniform,
1, false, uvTransform, 0);
                    GLES20.glUniform1f(depthAspectRatioUniform,
depthAspectRatio);
        }

        // Set the vertex attributes.
                    GLES20.glBindBuffer(GLES20.GL_ARRAY_BUFFER,
vertexBufferId);

        GLES20.glVertexAttribPointer(
                        positionAttribute,  COORDS_PER_VERTEX,
GLES20.GL_FLOAT, false, 0, verticesBaseAddress);
              GLES20.glVertexAttribPointer(normalAttribute,  3,
GLES20.GL_FLOAT, false, 0, normalsBaseAddress);
        GLES20.glVertexAttribPointer(
                texCoordAttribute, 2, GLES20.GL_FLOAT, false,
0, texCoordsBaseAddress);

        GLES20.glBindBuffer(GLES20.GL_ARRAY_BUFFER, 0);

        // Set the ModelViewProjection matrix in the shader.
          GLES20.glUniformMatrix4fv(modelViewUniform, 1, false,
modelViewMatrix, 0);
           GLES20.glUniformMatrix4fv(modelViewProjectionUniform,
1, false, modelViewProjectionMatrix, 0);

        // Enable vertex arrays
        GLES20.glEnableVertexAttribArray(positionAttribute);
        GLES20.glEnableVertexAttribArray(normalAttribute);
```

```
        GLES20.glEnableVertexAttribArray(texCoordAttribute);

        if (blendMode != null) {
            GLES20.glEnable(GLES20.GL_BLEND);
            switch (blendMode) {
                case Shadow:
                        // Multiplicative blending function for
Shadow.
                    GLES20.glDepthMask(false);
                            GLES20.glBlendFunc(GLES20.GL_ZERO,
GLES20.GL_ONE_MINUS_SRC_ALPHA);
                    break;
                case AlphaBlending:
                      // Alpha blending function, with the depth
mask enabled.
                    GLES20.glDepthMask(true);

                      // Textures are loaded with premultiplied
alpha
                                                              //
(https://developer.android.com/reference/android/graphics/Bitm
apFactory.Options#inPremultiplied),
                    // so we use the premultiplied alpha blend
factors.
                                GLES20.glBlendFunc(GLES20.GL_ONE,
GLES20.GL_ONE_MINUS_SRC_ALPHA);
                    break;
            }
        }

             GLES20.glBindBuffer(GLES20.GL_ELEMENT_ARRAY_BUFFER,
indexBufferId);
```

```
            GLES20.glDrawElements(GLES20.GL_TRIANGLES, indexCount,
GLES20.GL_UNSIGNED_SHORT, 0);
        GLES20.glBindBuffer(GLES20.GL_ELEMENT_ARRAY_BUFFER, 0);

        if (blendMode != null) {
            GLES20.glDisable(GLES20.GL_BLEND);
            GLES20.glDepthMask(true);
        }

        // Disable vertex arrays
        GLES20.glDisableVertexAttribArray(positionAttribute);
        GLES20.glDisableVertexAttribArray(normalAttribute);
        GLES20.glDisableVertexAttribArray(texCoordAttribute);

        GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, 0);

        ShaderUtil.checkGLError(TAG, "After draw");
    }

    private static void normalizeVec3(float[] v) {
            float reciprocalLength = 1.0f / (float) Math.sqrt(v[0]
* v[0] + v[1] * v[1] + v[2] * v[2]);
        v[0] *= reciprocalLength;
        v[1] *= reciprocalLength;
        v[2] *= reciprocalLength;
    }

    public void setUvTransformMatrix(float[] transform) {
        uvTransform = transform;
    }
```

```
    public void setDepthTexture(int textureId, int width, int
height) {
        depthTextureId = textureId;
        depthAspectRatio = (float) width / (float) height;
    }
}

package com.maquiAR.arface.rendering;

import android.content.Context;
import android.opengl.GLES20;
import android.util.Log;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Map;
import java.util.TreeMap;

public class ShaderUtil {

    /**
     * Converts a raw text file, saved as a resource, into an
OpenGL ES shader.
     *
     * @param type The type of shader we will be creating.
     * @param filename The filename of the asset file about to
be turned into a shader.
     * @param defineValuesMap The #define values to add to the
top of the shader source code.
     * @return The shader object handler.
```

```
    */
   public static int loadGLShader(
                String tag, Context context, int type, String
filename, Map<String, Integer> defineValuesMap)
            throws IOException {
        // Load shader source code.
              String code = readShaderFileFromAssets(context,
filename);

          // Prepend any #define values specified during this
run.
        String defines = "";
                  for (Map.Entry<String, Integer> entry :
defineValuesMap.entrySet()) {
               defines += "#define " + entry.getKey() + " " +
entry.getValue() + "\n";
        }
        code = defines + code;

        // Compiles shader code.
        int shader = GLES20.glCreateShader(type);
        GLES20.glShaderSource(shader, code);
        GLES20.glCompileShader(shader);

        // Get the compilation status.
        final int[] compileStatus = new int[1];
          GLES20.glGetShaderiv(shader, GLES20.GL_COMPILE_STATUS,
compileStatus, 0);

        // If the compilation failed, delete the shader.
        if (compileStatus[0] == 0) {
```

```
                    Log.e(tag, "Error compiling shader: " +
GLES20.glGetShaderInfoLog(shader));
            GLES20.glDeleteShader(shader);
            shader = 0;
        }

        if (shader == 0) {
                    throw new RuntimeException("Error creating
shader.");
        }

        return shader;
    }

     /** Overload of loadGLShader that assumes no additional
#define values to add. */
    public static int loadGLShader(String tag, Context context,
int type, String filename)
            throws IOException {
              Map<String, Integer> emptyDefineValuesMap = new
TreeMap<>();
             return loadGLShader(tag, context, type, filename,
emptyDefineValuesMap);
    }

    /**
     * Checks if we've had an error inside of OpenGL ES, and if
so what that error is.
     *
     * @param label Label to report in case of error.
     * @throws RuntimeException If an OpenGL error is detected.
     */
```

```
    public static void checkGLError(String tag, String label) {
        int lastError = GLES20.GL_NO_ERROR;
        // Drain the queue of all errors.
        int error;
                    while   ((error   =   GLES20.glGetError())   !=
GLES20.GL_NO_ERROR) {
            Log.e(tag, label + ": glError " + error);
            lastError = error;
        }
        if (lastError != GLES20.GL_NO_ERROR) {
              throw new RuntimeException(label + ": glError " +
lastError);
        }
    }

    /**
     * Converts a raw shader file into a string.
     *
     * @param filename The filename of the shader file about to
be turned into a shader.
     * @return The context of the text file, or null in case of
error.
     */
      private  static  String  readShaderFileFromAssets(Context
context, String filename)
            throws IOException {
                        try    (InputStream    inputStream    =
context.getAssets().open(filename);
                BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream))) {
            StringBuilder sb = new StringBuilder();
            String line;
```

```
            while ((line = reader.readLine()) != null) {
                String[] tokens = line.split(" ", -1);
                if (tokens[0].equals("#include")) {
                    String includeFilename = tokens[1];
                                                includeFilename  =
includeFilename.replace("\"", "");
                    if (includeFilename.equals(filename)) {
                           throw new IOException("Do not include
the calling file.");
                    }
                     sb.append(readShaderFileFromAssets(context,
includeFilename));
                } else {
                    sb.append(line).append("\n");
                }
            }
            return sb.toString();
        }
    }
}
/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed  under  the  Apache  License,  Version  2.0  (the
"License");
* you  may  not  use  this  file  except  in  compliance  with  the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
```

```
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
*  See  the  License  for  the  specific  language  governing
permissions and
* limitations under the License.
*/

package com.maquiAR.arface.helpers;

import android.content.Context;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraManager;
import android.hardware.display.DisplayManager;
import android.view.Display;
import android.view.Surface;
import android.view.WindowManager;
import com.google.ar.core.Session;

/**
* Helper  to  track  the  display  rotations.  In  particular,  the
180 degree rotations are not notified
*  by  the  onSurfaceChanged()  callback,  and  thus  they  require
listening to the android display
* events.
*/
public    final    class    DisplayRotationHelper    implements
DisplayManager.DisplayListener {
   private boolean viewportChanged;
   private int viewportWidth;
```

```
    private int viewportHeight;
    private final Display display;
    private final DisplayManager displayManager;
    private final CameraManager cameraManager;

    /**
     * Constructs the DisplayRotationHelper but does not
register the listener yet.
     *
     * @param context the Android {@link Context}.
     */
    public DisplayRotationHelper(Context context) {
                        displayManager = (DisplayManager)
context.getSystemService(Context.DISPLAY_SERVICE);
                         cameraManager = (CameraManager)
context.getSystemService(Context.CAMERA_SERVICE);
              WindowManager windowManager = (WindowManager)
context.getSystemService(Context.WINDOW_SERVICE);
        display = windowManager.getDefaultDisplay();
    }

    /** Registers the display listener. Should be called from
{@link Activity#onResume()}. */
    public void onResume() {
        displayManager.registerDisplayListener(this, null);
    }

    /** Unregisters the display listener. Should be called from
{@link Activity#onPause()}. */
    public void onPause() {
        displayManager.unregisterDisplayListener(this);
    }
```

```
    /**
     * Records a change in surface dimensions. This will be
later used by {@link
     * #updateSessionIfNeeded(Session)}. Should be called from
{@link
     * android.opengl.GLSurfaceView.Renderer
                                                              *
#onSurfaceChanged(javax.microedition.khronos.opengles.GL10,
int, int)}.
     *
     * @param width the updated width of the surface.
     * @param height the updated height of the surface.
     */
    public void onSurfaceChanged(int width, int height) {
        viewportWidth = width;
        viewportHeight = height;
        viewportChanged = true;
    }

    /**
     * Updates the session display geometry if a change was
posted either by {@link
         * #onSurfaceChanged(int, int)} call or by {@link
#onDisplayChanged(int)} system callback. This
     * function should be called explicitly before each call to
{@link Session#update()}. This
         * function will also clear the 'pending update'
(viewportChanged) flag.
     *
     * @param session the {@link Session} object to update if
display geometry changed.
```

```
     */
    public void updateSessionIfNeeded(Session session) {
        if (viewportChanged) {
            int displayRotation = display.getRotation();
                    session.setDisplayGeometry(displayRotation,
viewportWidth, viewportHeight);
            viewportChanged = false;
        }
    }

    /**
       *  Returns the aspect ratio of the GL surface viewport
while accounting for the display rotation
     *  relative to the device camera sensor orientation.
     */
                                          public             float
getCameraSensorRelativeViewportAspectRatio(String cameraId) {
        float aspectRatio;
                        int   cameraSensorToDisplayRotation   =
getCameraSensorToDisplayRotation(cameraId);
        switch (cameraSensorToDisplayRotation) {
            case 90:
            case 270:
                 aspectRatio = (float) viewportHeight / (float)
viewportWidth;
                break;
            case 0:
            case 180:
                  aspectRatio = (float) viewportWidth / (float)
viewportHeight;
                break;
            default:
```

```
                throw new RuntimeException("Unhandled rotation:
" + cameraSensorToDisplayRotation);
        }
        return aspectRatio;
    }

    /**
     * Returns the rotation of the back-facing camera with
respect to the display. The value is one of
     * 0, 90, 180, 270.
     */
        public int getCameraSensorToDisplayRotation(String
cameraId) {
        CameraCharacteristics characteristics;
        try {
                                            characteristics =
cameraManager.getCameraCharacteristics(cameraId);
        } catch (CameraAccessException e) {
                throw new RuntimeException("Unable to determine
display orientation", e);
        }

        // Camera sensor orientation.
                                    int sensorOrientation =
characteristics.get(CameraCharacteristics.SENSOR_ORIENTATION);

        // Current display orientation.
                                    int displayOrientation =
toDegrees(display.getRotation());

        // Make sure we return 0, 90, 180, or 270 degrees.
```

```
        return (sensorOrientation - displayOrientation + 360) %
360;
    }

    private int toDegrees(int rotation) {
        switch (rotation) {
            case Surface.ROTATION_0:
                return 0;
            case Surface.ROTATION_90:
                return 90;
            case Surface.ROTATION_180:
                return 180;
            case Surface.ROTATION_270:
                return 270;
            default:
                  throw new RuntimeException("Unknown rotation "
+ rotation);
        }
    }

    @Override
    public void onDisplayAdded(int displayId) {}

    @Override
    public void onDisplayRemoved(int displayId) {}

    @Override
    public void onDisplayChanged(int displayId) {
        viewportChanged = true;
    }
}
```

```
/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed under the Apache License, Version 2.0 (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
*  See the License for the specific language governing
permissions and
* limitations under the License.
*/
package com.maquiAR.arface.helpers;

import android.app.Activity;
import android.view.View;
import android.widget.TextView;

import
com.google.android.material.snackbar.BaseTransientBottomBar;
import com.google.android.material.snackbar.Snackbar;

/**
```

```
* Helper to manage the sample snackbar. Hides the Android
boilerplate code, and exposes simpler
* methods.
*/
public final class SnackbarHelper {
   private static final int BACKGROUND_COLOR = 0xbf323232;
   private Snackbar messageSnackbar;
   private enum DismissBehavior { HIDE, SHOW, FINISH };
   private int maxLines = 2;
   private String lastMessage = "";
   private View snackbarView;

   public boolean isShowing() {
       return messageSnackbar != null;
   }

   /** Shows a snackbar with a given message. */
    public void showMessage(Activity activity, String message)
{
               if (!message.isEmpty() && (!isShowing() ||
!lastMessage.equals(message))) {
           lastMessage = message;
           show(activity, message, DismissBehavior.HIDE);
       }
   }

    /** Shows a snackbar with a given message, and a dismiss
button. */
      public void showMessageWithDismiss(Activity activity,
String message) {
       show(activity, message, DismissBehavior.SHOW);
   }
```

```
    /**
     * Shows a snackbar with a given error message. When dismissed, will finish the activity. Useful
     * for notifying errors, where no further interaction with the activity is possible.
     */
    public void showError(Activity activity, String errorMessage) {
        show(activity, errorMessage, DismissBehavior.FINISH);
    }

    /**
     * Hides the currently showing snackbar, if there is one. Safe to call from any thread. Safe to
     * call even if snackbar is not shown.
     */
    public void hide(Activity activity) {
        if (!isShowing()) {
            return;
        }
        lastMessage = "";
        Snackbar messageSnackbarToHide = messageSnackbar;
        messageSnackbar = null;
        activity.runOnUiThread(
                new Runnable() {
                    @Override
                    public void run() {
                        messageSnackbarToHide.dismiss();
                    }
                });
    }
```

```
    public void setMaxLines(int lines) {
        maxLines = lines;
    }

    /**
     * Sets the view that will be used to find a suitable parent view to hold the Snackbar view.
     *
     * <p>To use the root layout ({@link android.R.id.content}), pass in {@code null}.
     *
     * @param snackbarView the view to pass to {@link com.google.android.material.snackbar.Snackbar#make(…)} which will be used to find a
     *                     suitable parent, which is a {@link androidx.coordinatorlayout.widget.CoordinatorLayout}, or
     *                     the window decor's content view, whichever comes first.
     */
    public void setParentView(View snackbarView) {
        this.snackbarView = snackbarView;
    }

    private void show(
            final Activity activity, final String message, final DismissBehavior dismissBehavior) {
        activity.runOnUiThread(
                new Runnable() {
                    @Override
                    public void run() {
```

```
                messageSnackbar =
                        Snackbar.make(
                                snackbarView == null
                                                ?
activity.findViewById(android.R.id.content)
                                                : snackbarView,
                                message,

Snackbar.LENGTH_INDEFINITE);

messageSnackbar.getView().setBackgroundColor(BACKGROUND_COLOR)
;
                                 if (dismissBehavior !=
DismissBehavior.HIDE) {
                        messageSnackbar.setAction(
                                "Dismiss",
                                  new View.OnClickListener()
{
                                    @Override
                                                public void
onClick(View v) {

messageSnackbar.dismiss();
                                    }
                                });
                                 if (dismissBehavior ==
DismissBehavior.FINISH) {
                                messageSnackbar.addCallback(
                                                new
BaseTransientBottomBar.BaseCallback<Snackbar>() {
                                        @Override
```

```
                                                    public void
onDismissed(Snackbar transientBottomBar, int event) {

super.onDismissed(transientBottomBar, event);

activity.finish();
                                                        }
                                                    });
                                }
                            }
                            ((TextView)
                                    messageSnackbar
                                            .getView()

.findViewById(com.google.android.material.R.id.snackbar_text))
                                    .setMaxLines(maxLines);
                            messageSnackbar.show();
                        }
                    });
    }
}

/*
* Copyright 2019 Google Inc. All Rights Reserved.
*  Licensed  under  the  Apache  License,  Version  2.0  (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
```

```
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.maquiAR.arface.helpers;

import android.app.Activity;
import android.view.WindowManager;

import com.google.ar.core.Camera;
import com.google.ar.core.TrackingFailureReason;
import com.google.ar.core.TrackingState;

/** Gets human readibly tracking failure reasons and suggested actions. */
public final class TrackingStateHelper {
   private static final String INSUFFICIENT_FEATURES_MESSAGE =
           "Can't find anything. Aim device at a surface with more texture or color.";
    private static final String EXCESSIVE_MOTION_MESSAGE = "Moving too fast. Slow down.";
   private static final String INSUFFICIENT_LIGHT_MESSAGE =
           "Too dark. Try moving to a well-lit area.";
   private static final String BAD_STATE_MESSAGE =
           "Tracking lost due to bad internal state. Please try restarting the AR experience.";
```

```
    private static final String CAMERA_UNAVAILABLE_MESSAGE =
              "Another app is using the camera. Tap on this app
or try closing the other one.";

    private final Activity activity;

    private TrackingState previousTrackingState;

    public TrackingStateHelper(Activity activity) {
        this.activity = activity;
    }

     /** Keep the screen unlocked while tracking, but allow it
to lock when tracking stops. */
          public    void    updateKeepScreenOnFlag(TrackingState
trackingState) {
        if (trackingState == previousTrackingState) {
            return;
        }

        previousTrackingState = trackingState;
        switch (trackingState) {
            case PAUSED:
            case STOPPED:
                activity.runOnUiThread(
                                                                ()    ->
activity.getWindow().clearFlags(WindowManager.LayoutParams.FLA
G_KEEP_SCREEN_ON));
                break;
            case TRACKING:
                activity.runOnUiThread(
```

```
                                                                    () ->
activity.getWindow().addFlags(WindowManager.LayoutParams.FLAG_
KEEP_SCREEN_ON));
                break;
        }
    }

    public static String getTrackingFailureReasonString(Camera
camera) {
                            TrackingFailureReason reason =
camera.getTrackingFailureReason();
        switch (reason) {
            case NONE:
                return "";
            case BAD_STATE:
                return BAD_STATE_MESSAGE;
            case INSUFFICIENT_LIGHT:
                return INSUFFICIENT_LIGHT_MESSAGE;
            case EXCESSIVE_MOTION:
                return EXCESSIVE_MOTION_MESSAGE;
            case INSUFFICIENT_FEATURES:
                return INSUFFICIENT_FEATURES_MESSAGE;
            case CAMERA_UNAVAILABLE:
                return CAMERA_UNAVAILABLE_MESSAGE;
        }
        return "Unknown tracking failure reason: " + reason;
    }
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
```

```
/*
* Licensed under the Apache License, Version 2.0 (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
* See the License for the specific language governing
permissions and
* limitations under the License.
*/

precision mediump float;

uniform sampler2D u_Texture;

uniform vec4 u_LightingParameters;
uniform vec4 u_MaterialParameters;
uniform vec4 u_ColorCorrectionParameters;

#if USE_DEPTH_FOR_OCCLUSION
uniform sampler2D u_DepthTexture;
uniform mat3 u_DepthUvTransform;
uniform float u_DepthAspectRatio;
#endif // USE_DEPTH_FOR_OCCLUSION
```

```
varying vec3 v_ViewPosition;
varying vec3 v_ViewNormal;
varying vec2 v_TexCoord;
varying vec3 v_ScreenSpacePosition;
uniform vec4 u_ObjColor;

#if USE_DEPTH_FOR_OCCLUSION

float DepthGetMillimeters(in sampler2D depth_texture, in vec2
depth_uv) {
 // Depth is packed into the red and green components of its
texture.
 // The texture is a normalized format, storing millimeters.
  vec3 packedDepthAndVisibility = texture2D(depth_texture,
depth_uv).xyz;
 return dot(packedDepthAndVisibility.xy, vec2(255.0, 256.0 *
255.0));
}

// Returns linear interpolation position of value between min
and max bounds.
// E.g., DepthInverseLerp(1100, 1000, 2000) returns 0.1.
float DepthInverseLerp(in float value, in float min_bound, in
float max_bound) {
 return clamp((value - min_bound) / (max_bound - min_bound),
0.0, 1.0);
}

// Returns a value between 0.0 (not visible) and 1.0
(completely visible)
```

```
// Which represents how visible or occluded is the pixel in relation to the
// depth map.
float DepthGetVisibility(in sampler2D depth_texture, in vec2 depth_uv,
                         in float asset_depth_mm) {
    float depth_mm = DepthGetMillimeters(depth_texture, depth_uv);

  // Instead of a hard z-buffer test, allow the asset to fade into the
   // background along a 2 * kDepthTolerancePerMm * asset_depth_mm
  // range centered on the background depth.
  const float kDepthTolerancePerMm = 0.015;
   float visibility_occlusion = clamp(0.5 * (depth_mm - asset_depth_mm) /
    (kDepthTolerancePerMm * asset_depth_mm) + 0.5, 0.0, 1.0);

  // Depth close to zero is most likely invalid, do not use it for occlusions.
  float visibility_depth_near = 1.0 - DepthInverseLerp(
                            depth_mm,     /*min_depth_mm=*/150.0,
/*max_depth_mm=*/200.0);

  // Same for very high depth values.
  float visibility_depth_far = DepthInverseLerp(
                           depth_mm,     /*min_depth_mm=*/7500.0,
/*max_depth_mm=*/8000.0);

  const float kOcclusionAlpha = 0.0;
  float visibility =
```

```
    max(max(visibility_occlusion, kOcclusionAlpha),
        max(visibility_depth_near, visibility_depth_far));

 return visibility;
}

float DepthGetBlurredVisibilityAroundUV(in sampler2D
depth_texture, in vec2 uv,
                                        in float
asset_depth_mm) {
 // Kernel used:
 // 0   4   7   4   0
 // 4   16  26  16  4
 // 7   26  41  26  7
 // 4   16  26  16  4
 // 0   4   7   4   0
 const float kKernelTotalWeights = 269.0;
 float sum = 0.0;

 const float kOcclusionBlurAmount = 0.01;
 vec2 blurriness = vec2(kOcclusionBlurAmount,
                        kOcclusionBlurAmount *
u_DepthAspectRatio);

 float current = 0.0;

 current += DepthGetVisibility(depth_texture, uv + vec2(-1.0,
-2.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+1.0,
-2.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(-1.0,
+2.0) * blurriness, asset_depth_mm);
```

```
 current += DepthGetVisibility(depth_texture, uv + vec2(+1.0,
+2.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(-2.0,
+1.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+2.0,
+1.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(-2.0,
-1.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+2.0,
-1.0) * blurriness, asset_depth_mm);
 sum += current * 4.0;

 current = 0.0;
 current += DepthGetVisibility(depth_texture, uv + vec2(-2.0,
-0.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+2.0,
+0.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+0.0,
+2.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(-0.0,
-2.0) * blurriness, asset_depth_mm);
 sum += current * 7.0;

 current = 0.0;
 current += DepthGetVisibility(depth_texture, uv + vec2(-1.0,
-1.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+1.0,
-1.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(-1.0,
+1.0) * blurriness, asset_depth_mm);
 current += DepthGetVisibility(depth_texture, uv + vec2(+1.0,
+1.0) * blurriness, asset_depth_mm);
```

```
 sum += current * 16.0;

 current = 0.0;
  current += DepthGetVisibility(depth_texture, uv + vec2(+0.0,
+1.0) * blurriness, asset_depth_mm);
  current += DepthGetVisibility(depth_texture, uv + vec2(-0.0,
-1.0) * blurriness, asset_depth_mm);
  current += DepthGetVisibility(depth_texture, uv + vec2(-1.0,
-0.0) * blurriness, asset_depth_mm);
  current += DepthGetVisibility(depth_texture, uv + vec2(+1.0,
+0.0) * blurriness, asset_depth_mm);
 sum += current * 26.0;

 sum += DepthGetVisibility(depth_texture, uv , asset_depth_mm)
* 41.0;

 return sum / kKernelTotalWeights;
}

#endif // USE_DEPTH_FOR_OCCLUSION

void main() {
   // We support approximate sRGB gamma.
   const float kGamma = 0.4545454;
   const float kInverseGamma = 2.2;
   const float kMiddleGrayGamma = 0.466;

     // Unpack lighting and material parameters for better
naming.
   vec3 viewLightDirection = u_LightingParameters.xyz;
   vec3 colorShift = u_ColorCorrectionParameters.rgb;
```

```
                    float       averagePixelIntensity       =
u_ColorCorrectionParameters.a;

   float materialAmbient = u_MaterialParameters.x;
   float materialDiffuse = u_MaterialParameters.y;
   float materialSpecular = u_MaterialParameters.z;
   float materialSpecularPower = u_MaterialParameters.w;

    // Normalize varying parameters, because they are linearly
interpolated in the vertex shader.
   vec3 viewFragmentDirection = normalize(v_ViewPosition);
   vec3 viewNormal = normalize(v_ViewNormal);

    // Flip the y-texture coordinate to address the texture
from top-left.
    vec4 objectColor = texture2D(u_Texture, vec2(v_TexCoord.x,
1.0 - v_TexCoord.y));

    // Apply color to grayscale image only if the alpha of
u_ObjColor is
   // greater and equal to 255.0.
   objectColor.rgb *= mix(vec3(1.0), u_ObjColor.rgb / 255.0,
                          step(255.0, u_ObjColor.a));

    // Apply inverse SRGB gamma to the texture before making
lighting calculations.
                objectColor.rgb     =     pow(objectColor.rgb,
vec3(kInverseGamma));

   // Ambient light is unaffected by the light intensity.
   float ambient = materialAmbient;
```

```
    // Approximate a hemisphere light (not a harsh directional
light).
   float diffuse = materialDiffuse *
           0.5 * (dot(viewNormal, viewLightDirection) + 1.0);

    // Compute specular light. Textures are loaded with with
premultiplied alpha
                                                              //
(https://developer.android.com/reference/android/graphics/Bitm
apFactory.Options#inPremultiplied),
   // so premultiply the specular color by alpha as well.
    vec3 reflectedLightDirection = reflect(viewLightDirection,
viewNormal);
                float     specularStrength     =     max(0.0,
dot(viewFragmentDirection, reflectedLightDirection));
   float specular = objectColor.a * materialSpecular *
           pow(specularStrength, materialSpecularPower);

     vec3 color = objectColor.rgb * (ambient + diffuse) +
specular;
   // Apply SRGB gamma before writing the fragment color.
   color.rgb = pow(color, vec3(kGamma));
   // Apply average pixel intensity and color shift
        color   *=   colorShift   *   (averagePixelIntensity   /
kMiddleGrayGamma);
   gl_FragColor.rgb = color;
   gl_FragColor.a = objectColor.a;

#if USE_DEPTH_FOR_OCCLUSION
   const float kMetersToMillimeters = 1000.0;
           float    asset_depth_mm    =    v_ViewPosition.z    *
kMetersToMillimeters * -1.;
```

```
    // Computes the texture coordinates to sample from the
depth image.
            vec2    depth_uvs    =    (u_DepthUvTransform    *
vec3(v_ScreenSpacePosition.xy, 1)).xy;

   // The following step is very costly. Replace the last line
with the
   // commented line if it's too expensive.
      //  gl_FragColor  *=  DepthGetVisibility(u_DepthTexture,
depth_uvs, asset_depth_mm);
                                        gl_FragColor            *=
DepthGetBlurredVisibilityAroundUV(u_DepthTexture,    depth_uvs,
asset_depth_mm);
#endif // USE_DEPTH_FOR_OCCLUSION
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed  under  the  Apache  License,  Version  2.0  (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
```

```
* See the License for the specific language governing permissions and
* limitations under the License.
*/

uniform mat4 u_ModelView;
uniform mat4 u_ModelViewProjection;

attribute vec4 a_Position;
attribute vec3 a_Normal;
attribute vec2 a_TexCoord;

varying vec3 v_ViewPosition;
varying vec3 v_ViewNormal;
varying vec2 v_TexCoord;
varying vec3 v_ScreenSpacePosition;

void main() {
   v_ViewPosition = (u_ModelView * a_Position).xyz;
      v_ViewNormal = normalize((u_ModelView * vec4(a_Normal, 0.0)).xyz);
   v_TexCoord = a_TexCoord;
   gl_Position = u_ModelViewProjection * a_Position;
   v_ScreenSpacePosition = gl_Position.xyz / gl_Position.w;
}

/*
* Copyright 2020 Google Inc. All Rights Reserved.
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
```

```
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

precision mediump float;

uniform sampler2D u_DepthTexture;

varying vec2 v_TexCoord;

const highp float kMaxDepth = 8000.0; // In millimeters.

float DepthGetMillimeters(in sampler2D depth_texture, in vec2 depth_uv) {
 // Depth is packed into the red and green components of its texture.
 // The texture is a normalized format, storing millimeters.
  vec3 packedDepthAndVisibility = texture2D(depth_texture, depth_uv).xyz;
 return dot(packedDepthAndVisibility.xy, vec2(255.0, 256.0 * 255.0));
```

```
}

// Returns a color corresponding to the depth passed in. Colors range from red
// to green to blue, where red is closest and blue is farthest.
//
// Uses Turbo color mapping:
//
https://ai.googleblog.com/2019/08/turbo-improved-rainbow-colormap-for.html
vec3 DepthGetColorVisualization(in float x) {
   const vec4 kRedVec4 = vec4(0.55305649, 3.00913185, -5.46192616, -11.11819092);
   const vec4 kGreenVec4 = vec4(0.16207513, 0.17712472, 15.24091500, -36.50657960);
   const vec4 kBlueVec4 = vec4(-0.05195877, 5.18000081, -30.94853351, 81.96403246);
 const vec2 kRedVec2 = vec2(27.81927491, -14.87899417);
 const vec2 kGreenVec2 = vec2(25.95549545, -5.02738237);
 const vec2 kBlueVec2 = vec2(-86.53476570, 30.23299484);
 const float kInvalidDepthThreshold = 0.01;

  // Adjusts color space via 6 degree poly interpolation to avoid pure red.
 x = clamp(x * 0.9 + 0.03, 0.0, 1.0);
 vec4 v4 = vec4(1.0, x, x * x, x * x * x);
 vec2 v2 = v4.zw * v4.z;
 vec3 polynomial_color = vec3(
   dot(v4, kRedVec4) + dot(v2, kRedVec2),
   dot(v4, kGreenVec4) + dot(v2, kGreenVec2),
   dot(v4, kBlueVec4) + dot(v2, kBlueVec2)
```

```
 );

 return step(kInvalidDepthThreshold, x) * polynomial_color;
}

void main() {
 highp float normalized_depth =
       clamp(DepthGetMillimeters(u_DepthTexture, v_TexCoord.xy)
/ kMaxDepth,
           0.0, 1.0);
                vec4                depth_color                =
vec4(DepthGetColorVisualization(normalized_depth), 1.0);
 gl_FragColor = depth_color;
}

/*
* Copyright 2020 Google Inc. All Rights Reserved.
*  Licensed  under  the  Apache  License,  Version  2.0  (the
"License");
*  you  may  not  use  this  file  except  in  compliance  with  the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
```

```
* See the License for the specific language governing permissions and
* limitations under the License.
*/

attribute vec4 a_Position;
attribute vec2 a_TexCoord;

varying vec2 v_TexCoord;

void main() {
  v_TexCoord = a_TexCoord;
  gl_Position = a_Position;
}

precision mediump float;

uniform sampler2D u_Texture;
uniform sampler2D u_TextureEye;

uniform vec4 u_LightingParameters;
uniform vec4 u_MaterialParameters;
uniform vec4 u_ColorCorrectionParameters;

varying vec3 v_ViewPosition;
varying vec3 v_ViewNormal;
varying vec2 v_TexCoord;
uniform vec4 u_ObjColor;
uniform vec4 u_TintColor;
uniform vec4 u_TintColorEyes;

void main() {
```

```
   // We support approximate sRGB gamma.
   const float kGamma = 0.4545454;
   const float kInverseGamma = 2.2;
   const float kMiddleGrayGamma = 0.466;

     // Unpack lighting and material parameters for better
naming.
   vec3 viewLightDirection = u_LightingParameters.xyz;
   vec3 colorShift = u_ColorCorrectionParameters.rgb;
                     float       averagePixelIntensity       =
u_ColorCorrectionParameters.a;

   float materialAmbient = u_MaterialParameters.x;
   float materialDiffuse = u_MaterialParameters.y;
   float materialSpecular = u_MaterialParameters.z;
   float materialSpecularPower = u_MaterialParameters.w;

    // Normalize varying parameters, because they are linearly
interpolated in the vertex shader.
   vec3 viewFragmentDirection = normalize(v_ViewPosition);
   vec3 viewNormal = normalize(v_ViewNormal);

     // Flip the y-texture coordinate to address the texture
from top-left.
          vec4   objectColorContour   =   texture2D(u_Texture,
vec2(v_TexCoord.x, 1.0 - v_TexCoord.y));
          vec4   objectColorEye   =   texture2D(u_TextureEye,
vec2(v_TexCoord.x, 1.0 - v_TexCoord.y));

        objectColorContour.rgb   =   objectColorContour.rgb   *
u_TintColor.rgb;
```

```
            objectColorEye.rgb    =    objectColorEye.rgb    *
u_TintColorEyes.rgb;
   vec4 objectColor = objectColorContour + objectColorEye;

    // Apply color to grayscale image only if the alpha of
u_ObjColor is
   // greater and equal to 255.0.
   objectColor.rgb *= mix(vec3(1.0), u_ObjColor.rgb / 255.0,
                          step(255.0, u_ObjColor.a));

    // Apply inverse SRGB gamma to the texture before making
lighting calculations.
               objectColor.rgb     =     pow(objectColor.rgb,
vec3(kInverseGamma));


   // Ambient light is unaffected by the light intensity.
   float ambient = materialAmbient;

    // Approximate a hemisphere light (not a harsh directional
light).
   float diffuse = materialDiffuse *
           0.5 * (dot(viewNormal, viewLightDirection) + 1.0);

   // Compute specular light.
    vec3 reflectedLightDirection = reflect(viewLightDirection,
viewNormal);
               float     specularStrength     =     max(0.0,
dot(viewFragmentDirection, reflectedLightDirection));
   float specular = materialSpecular *
           pow(specularStrength, materialSpecularPower);
```

```
     vec3 color = objectColor.rgb * (ambient + diffuse) +
specular;
   // Apply SRGB gamma before writing the fragment color.
   color.rgb = pow(color, vec3(kGamma));
   // Apply average pixel intensity and color shift
        color *= colorShift * (averagePixelIntensity /
kMiddleGrayGamma);
   gl_FragColor.rgb = color;
   gl_FragColor.a = objectColor.a;
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
* Licensed under the Apache License, Version 2.0 (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
* See the License for the specific language governing
permissions and
* limitations under the License.
*/
```

```
uniform mat4 u_ModelView;
uniform mat4 u_ModelViewProjection;

attribute vec4 a_Position;
attribute vec3 a_Normal;
attribute vec2 a_TexCoord;

varying vec3 v_ViewPosition;
varying vec3 v_ViewNormal;
varying vec2 v_TexCoord;

void main() {
   v_ViewPosition = (u_ModelView * a_Position).xyz;
      v_ViewNormal  =  normalize((u_ModelView  *  vec4(a_Normal,
0.0)).xyz);
   v_TexCoord = a_TexCoord;
   gl_Position = u_ModelViewProjection * a_Position;
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed  under  the  Apache  License,  Version  2.0  (the
"License");
* you may not use this file except in compliance with the
License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing,
software
```

```
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
*  See the License for the specific language governing permissions and
* limitations under the License.
*/
#extension GL_OES_EGL_image_external : require

precision mediump float;
varying vec2 v_TexCoord;
uniform samplerExternalOES sTexture;


void main() {
   gl_FragColor = texture2D(sTexture, v_TexCoord);
}

/*
* Copyright 2017 Google Inc. All Rights Reserved.
*  Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
```

```
* distributed under the License is distributed on an "AS IS"
BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
* See the License for the specific language governing
permissions and
* limitations under the License.
*/

attribute vec4 a_Position;
attribute vec2 a_TexCoord;

varying vec2 v_TexCoord;

void main() {
  gl_Position = a_Position;
  v_TexCoord = a_TexCoord;
}
```

```
plugins {
   id 'com.android.application'
   id 'kotlin-android'
   id 'kotlin-android-extensions'
}
```

Build.gradle

```
android {
   compileSdkVersion 30

   defaultConfig {
       applicationId "com.maquiAR"
       minSdkVersion 27
```

```
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

                                        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
                                                proguardFiles
getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility "1.8"
        targetCompatibility "1.8"
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {

    implementation 'com.google.ar:core:1.21.0'
    implementation 'de.javagl:obj:0.2.1'
                                                implementation
"org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.3.2'
```

```
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
                                                    implementation
'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
                                            androidTestImplementation
'androidx.test.espresso:espresso-core:3.3.0'
    implementation 'com.github.denzcoskun:ImageSlideshow:0.1.0'
    implementation 'com.google.code.gson:gson:2.8.2'
    implementation "androidx.preference:preference-ktx:1.1.1"

}
```

APÊNDICE C - Artigo sobre o Trabalho de Conclusão de Curso

MaquiAR: Solução com Realidade Aumentada aplicada no E-commerce de maquiagem

Mateus Nunes Cechetto[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

mateus.cechetto@grad.ufsc.br

Abstract. *Online shopping is increasingly popular. With the COVID-19 pandemic, both its advantages in relation to shopping in physical stores and its disadvantages were evidenced. One of the disadvantages is the impossibility to touch the product with your hands. The proposal of this work is an Android application that will allow users to try different types of makeup through the smartphone's camera, through filters with Augmented Reality and the users will have access to information about the makeup and how to buy them. The application development process, the architecture used and the difficulties encountered are described. Finally, tests performed with users and their results are presented.*

Resumo. *Compras online estão cada vez mais populares. Com a pandemia da COVID-19, foram evidenciados tanto suas vantagens em relação ao comércio em lojas físicas quanto suas desvantagens. Uma delas é a impossibilidade de tocar o produto com as mãos. A proposta deste trabalho é um aplicativo Android que permitirá os usuários experimentarem diferentes tipos de maquiagem através da câmera do smartphone, por meio de filtros com Realidade Aumentada e terá acesso a informações das maquiagens e de como comprá-las. É descrito o processo de desenvolvimento do aplicativo, a arquitetura utilizada e as dificuldades encontradas. Por fim, são apresentados testes realizados com usuários e os seus resultados.*

1. Introdução

E-commerce, também conhecido como comércio eletrônico, "refere-se a um modelo de negócio que permite empresas e pessoas comprarem e venderem produtos e serviços pela internet". (BLOOMENTHAL, 2021 tradução nossa). No e-commerce, ao invés do comprador ir fisicamente à loja, olhar o produto, decidir se vai ou não comprá-lo, pagar, caso tenha optado em comprar, ele faz todo esse processo virtualmente. O e-commerce tem como uma de suas principais vantagens a comodidade que ele traz ao consumidor, pois funciona ininterruptamente, ao contrário de lojas físicas que possuem horário comercial, apresenta ao consumidor uma maior variedade de produtos e o consumidor não precisa sair de casa para realizar a compra. (BLOOMENTHAL, 2021)

O e-commerce, porém, também possui desvantagens. A segurança é uma das principais delas. (SAMPAIO, 2019). Tanto o consumidor necessita da certeza de que o site no qual está colocando seus dados é legítimo e seguro, quanto a empresa precisa da certeza de que o usuário é uma pessoa real, sem intenções de fraudes. Essa certeza é

necessária pois há inúmeras atividades fraudulentas na internet. Mais uma desvantagem é o tempo de entrega, (SAMPAIO, 2019) já que o comprador não sai com o produto em mãos como no comércio tradicional. Além disso, há 12 clientes que preferem "sentir" o produto com as próprias mãos; como por exemplo verificar o caimento ou a combinação de roupas, calçados e acessórios. Este é um problema que este trabalho irá tentar abordar, ou seja, a impossibilidade de visualizar um produto aplicado numa situação real.

Este trabalho apresenta uma proposta para abordar esse problema, através do uso de realidade aumentada (RA) na apresentação do produto ao consumidor, no caso deste trabalho, os produtos são maquiagens. Resumidamente, RA é a sobreposição de objetos virtuais ao mundo real do usuário. Ela se difere da realidade virtual, que tem como definição a criação de um mundo virtual no qual o usuário é inserido em tempo real, por meio de um avatar. Desta forma, a RA é mais benéfica do que a realidade virtual no contexto de e-commerce porque ela melhora a compreensão dos consumidores sobre os produtos, e proporciona o prazer de se verem usando o item sem precisarem ir à loja física. (YIM; CHU; SAUER, 2017). Assim, o uso de RA no e-commerce promete melhorar a forma como um produto é apresentado ao consumidor e por conseguinte: melhorar a percepção e experiência do usuário em relação ao produto, aumentar a intenção de compra e diminuir as devoluções.

2. Estudos Realizados

2.1. Realidade Aumentada (RA)

De acordo com Azuma (1997 apud SILVA; ABREU; TEIXEIRA, 2021), realidade aumentada pode ser definida como uma variação da Realidade Virtual. Eles afirmam que a realidade virtual imerge o usuário completamente em um ambiente virtual, e o usuário, enquanto imerso, não consegue ver o mundo real. Em contrapartida, a RA permite ver o mundo real, com objetos virtuais sobrepostos e coexistindo com o mundo real. O autor completa dizendo que a RA complementa o mundo real, ao invés de substituí-lo, como faz a Realidade Virtual.

Lu e Smith (2007) definem realidade aumentada como uma tecnologia que consegue misturar ou sobrepor objetos virtuais ao mundo real. Ao contrário da Realidade Virtual que substitui o mundo físico por um virtual, a RA aprimora a realidade ao integrar objetos virtuais ao mundo físico. Esses objetos virtuais se tornam, de certa forma, uma parte igual do ambiente real. (LU; SMITH, 2007).

Segundo Carmigniani et al. (2011 apud SILVA; ABREU; TEIXEIRA, 2021), a RA pode ser vista como uma tecnologia interativa, sendo uma visão direta ou indireta de um ambiente físico que foi complementado pela adição de informações virtuais.

O objetivo da RA pode ser descrito como permitir ao seu utilizador uma interação com o ambiente real e com os elementos virtuais sobrepostos no mesmo, de uma forma natural e intuitiva, sem necessidade de uma grande adaptação por parte deste. (TORI; HOUNSELL, 2018 apud SILVA; ABREU; TEIXEIRA, 2021).

2.2. Virtual try-on

Virtual try-on, ou apenas try-on, é uma tecnologia que pode entregar ao 15 comprador informações de um produto semelhantes às informações obtidas pela experimentação direta do produto. Try-on permite aos consumidores ver os produtos por vários ângulos, dar zoom, rotacionar o produto e vê-lo em diferentes cores, em um modelo virtual que imita suas aparências. (KIM; FORSYTHE, 2008). Segundo Kim e Forsythe (2008) a interatividade e o envolvimento do consumidor criados pelo uso da tecnologia podem melhorar a experiência da compra online.

De acordo com Wagner (2007 apud KIM; FORSYTHE, 2008), virtual try-on está se tornando mais usado no comércio eletrônico de varejo pois diminui a diferença da experiência entre compras online e offline. O mesmo autor afirma ainda que adicionar tecnologias interativas como virtual try-on ao comércio eletrônico aumenta significantemente as taxas de conversão de um website.

Embora o conceito de try-on tenha sido criado antes de suas aplicações com realidade aumentada, é evidente que o uso da RA potencializa o try-on virtual de produtos, pois permite que, ao invés de utilizar modelos, o consumidor se veja utilizando o produto, melhorando ainda mais sua experiência.

2.3. Desenvolvimento de Realidade Aumentada com ARCore

ARCore é o software development kit (SDK) da Google voltado ao desenvolvimento de RA e foi lançado em março de 2018 como o sucessor do Projeto Tango, que tivera seu início em 2014. O kit possui suporte tanto para dispositivos Android quanto para dispositivos iOS. O ARCore é totalmente gratuito e de código aberto.

Como o objetivo deste trabalho é o desenvolvimento de um aplicativo de maquiagem, é preciso entender como as tecnologias de realidade aumentada funcionam para reconhecer e rastrear rostos. Assim, as maquiagens que serão desenvolvidas aparecerão na posição correta do rosto do usuário e acompanharão seus movimentos. Esta seção tem como foco as características do desenvolvimento com ARCore, tecnologia escolhida para o desenvolvimento do aplicativo.

O ARCore possui uma Application Programming Interface (API) específica para o desenvolvimento de aplicativos de Realidade Aumentada que desejam renderizar objetos virtuais, no caso deste trabalho os filtros de maquiagem, sobre rostos humanos. Esta API é chamada de Augmented Faces e fornece pontos de recurso que permitem que o aplicativo identifique automaticamente diferentes regiões de um rosto. (GOOGLE, 2022).

Um “rosto aumentado” é composto por três partes: um ponto central, três pontos de região e uma malha facial 3D. O ponto central localiza-se atrás do nariz e marca o meio da cabeça do usuário. É utilizado para para renderizar objetos como chapéus e bonés, portanto não muito relevante no contexto deste trabalho. Os pontos de região ficam localizados um na parte da esquerda da testa, um na direita, e outro na ponta do nariz e são utilizados para renderizar objetos no nariz e ao redor das orelhas. A malha facial 3D possui 468 pontos que permitem criar texturas adaptáveis e detalhadas que

seguem com precisão um rosto. É através dela que serão criados os filtros de maquiagens deste trabalho. (GOOGLE, 2022).

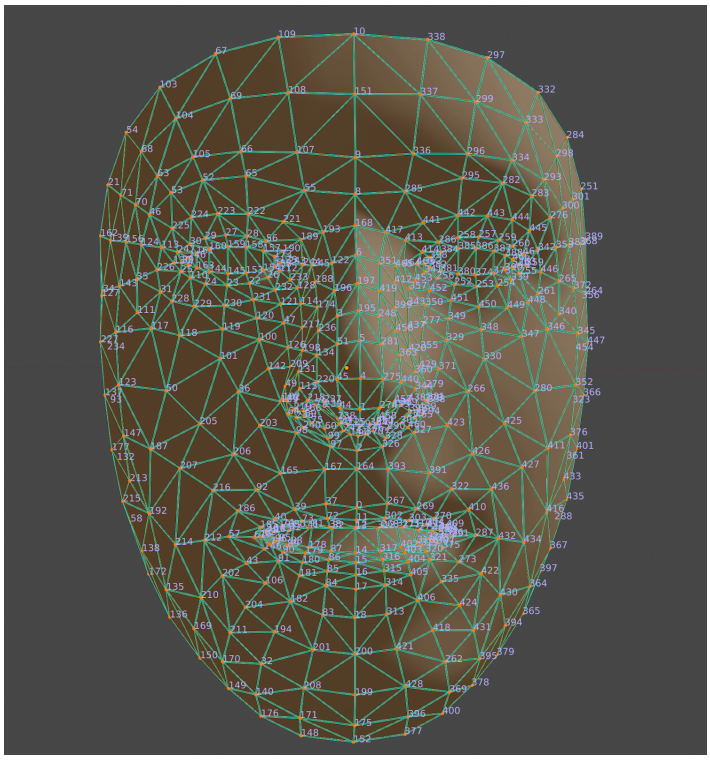


Figura 1. Malha facial 3D ARCore Augmented Faces API

3. Aplicativos semelhantes

Com o objetivo de analisar o estado atual do mercado, foi realizada uma pesquisa de aplicativos relacionados a try-on e e-commerce de maquiagem. Foram utilizados para a pesquisa os termos-chave: “try-on de maquiagem”, “experimentar maquiagem”, “maquiagem realidade aumentada”, “comprar maquiagem”, “e-commerce maquiagem” e seus sinônimos em inglês. A pesquisa foi feita na Google Play Store e foram selecionados os aplicativos que possuíam mais downloads. Foram selecionados 8 aplicativos que foram instalados para uma verificação se os mesmos de fato faziam parte do escopo do trabalho. Após a verificação, 3 destes aplicativos foram desconsiderados por serem jogos de maquiagem. Posteriormente, foram adicionados à lista de aplicativos a serem analisados aplicativos de marcas de cosméticos e beleza conhecidos, como o da “Sephora”, assim como os aplicativos “Amazon” e “Instagram”, que, apesar de não terem aparecido na pesquisa com termos-chave, são marketplaces de amplo uso no Brasil e que contém seções de maquiagem.

Os aplicativos foram separados em 4 categorias: aplicativos de e-commerce, aplicativos de try-on de maquiagem, aplicativos de e-commerce com try-on e aplicativos de try-on voltados ao e-commerce.

Aplicativo	Realidade Aumentada	Navegação entre produtos na tela de RA	E-commerce	Categoria	App Nacional

Época Cosméticos e Maquiagens	Não	Não aplicável	Sim	Aplicativo de e-commerce de maquiagem	Sim
Sephora	Não	Não aplicável	Sim	Aplicativo de e-commerce de maquiagem	Sim
Beleza na Web: Perfume | Cabelo	Não	Não aplicável	Sim	Aplicativo de e-commerce de maquiagem	Sim
Amazon (versão do Brasil)	Não	Não aplicável	Sim	Aplicativo de e-commerce de maquiagem	Versão Nacional
YouFace Makeup - Makeover Studio	Sim	Sim	Não	Aplicativo de try-on de maquiagem	Não
Instagram	Sim, em poucos produtos	Não	Direciona para o website da loja	Aplicativo de e-commerce com try-on de maquiagem	Versão Nacional
Amazon (versão estadunidense)	Sim	Entre cores do mesmo produto	Sim	Aplicativo de e-commerce com try-on de maquiagem	Não
Sephora - Beauty Shopping	Sim	Sim	Sim	Aplicativo de e-commerce com try-on de maquiagem	Não, e não está disponível no Brasil
MakeupPlus - Virtual Makeup	Sim	Sim	Direciona para o website da loja	Aplicativo de try-on de maquiagem voltado ao e-commerce	Não
YouCam Makeup - Selfie Editor	Sim	Sim	Direciona para o website da loja	Aplicativo de try-on de maquiagem voltado ao e-commerce	Não

Tabela 1. Comparativo entre os aplicativos analisados

A Tabela 1 faz a comparação entre os aplicativos analisados. Ela possui 6 colunas: “Aplicativo”, que é o nome do aplicativo analisado, “Realidade Aumentada”, que indica se o aplicativo utiliza realidade aumentada na apresentação do produto ao usuário, “Navegação entre produtos na tela de RA”, que indica se é possível experimentar diferentes produtos na tela de visualização com RA, “E-commerce”, que

indica se o aplicativo possui o objetivo de vender os produtos nele apresentados, a categoria que foram classificados e “App Nacional”, que indica se o aplicativo foi desenvolvido no Brasil ou possui uma versão brasileira.

Ao analisar a tabela, é possível encontrar algumas relações. A mais evidente é que aplicativos focados no e-commerce geralmente não possuem realidade aumentada. Outra é que aplicativos com o foco na realidade aumentada para o try-on de maquiagem direcionam os usuários para o site do produto, ao invés de ter o processo de e-commerce integrado. Apenas os aplicativos da Amazon (versão estadunidense) e Sephora - Beauty Shopping, possuem tanto o try-on quanto o e-commerce no próprio aplicativo, e ambos não estão disponíveis no Brasil. Outra relação é que os aplicativos nacionais não possuem realidade aumentada, nem a versão nacional do aplicativo da Amazon. Um fator que pode ter sido determinante para isso é a baixa quantidade de pessoas com dispositivos com capacidade para utilizar a tecnologia. Esse cenário, porém, vem mudando e é provável que estes aplicativos venham a aderir a realidade aumentada para exibir seus produtos nos próximos anos.

4. Proposta

Após realizar os estudos sobre realidade aumentada e seus impactos no processo de decisão de compra do consumidor e de analisar aplicativos populares relacionados à maquiagem, foi elaborada a solução proposta para este trabalho. Foram consideradas para a proposta, as funcionalidades encontradas pelos aplicativos analisados que o autor considerou mais úteis e impactantes. O sistema é um aplicativo móvel para o sistema operacional Android e se chama MaquiAR. O MaquiAR permitirá ao usuário experimentar diferentes tipos de maquiagens, através de filtros. Ao utilizar os filtros o usuário pode tirar fotos, compartilhar em redes sociais, salvar maquiagens em seus favoritos para vê-las depois. O usuário tem acesso a informações das maquiagens que estão sendo utilizadas no filtro que ele está usando, além de informações de como adquiri-las. O usuário pode, também, navegar pela lista de produtos disponíveis no aplicativo, além de visualizar seus favoritos.

A seguir é apresentado o posicionamento do MaquiAR na tabela utilizada para o comparativo dos aplicativos analisados anteriormente:

Aplicativo	**Realidade Aumentada**	**Navegação entre produtos na tela de RA**	**E-commerce**	**Categoria**	**App Nacional**
MaquiAR	Sim	Sim	Direciona para o website da loja	Aplicativo de try-on de maquiagem voltado ao e-commerce	Sim

Tabela 2. Posicionamento do MaquiAR na tabela de comparativo

O MaquiAR entra na categoria de aplicativos de try-on voltados ao e-commerce, pois o foco do aplicativo é nos filtros de maquiagem através da realidade aumentada e tem a visão de que ao experimentar maquiagens o usuário pode desejar comprá-las, então o aplicativo fornecerá ao usuário onde comprá-las. Aplicativos que direcionam

para as páginas das lojas possuem, em sua maioria, parcerias com as lojas, de tal forma que os aplicativos recebem das lojas ou uma porcentagem sobre as compras de clientes vindos dos links fornecidos por esses aplicativos ou um valor sobre cada cliente que acessou o link. As maquiagens do aplicativo serão genéricas e a funcionalidade de mostrar informações do produto e onde comprá-lo avisará o usuário que estão sendo utilizadas maquiagens genéricas, mostrará produtos reais que possuem características semelhantes à maquiagem genérica e avisará o usuário que o nível de fidelidade da maquiagem apresentada pode ter pequenas diferenças para os produtos.

5. Arquitetura

A maior parte deste trabalho foi desenvolvida usando o padrão MVC (Model-View-Controller) com exceção da parte relacionada à realidade aumentada, que foi separada num diretório a parte e suas especificidades que serão comentadas posteriormente. Não foi utilizado nenhum banco de dados nem servidor, apesar de ser totalmente aplicável. O motivo se dá pelo foco do trabalho ser na realidade aumentada e no desenvolvimento mobile Android.

Na camada Model ficam as entidades do domínio: Product, Blush, EyeShadow e Lipstick. Product é uma classe abstrata que define as características de um produto para o sistema: id, nome, descrição, imagens, ícone do tipo de maquiagem, link do site que o usuário será direcionado, textura que será usada na tela de realidade aumentada, vetor com os valores correspondentes às cores que serão aplicadas à textura, e o nome da cor. As classes Blush, EyeShadow e Lipstick são implementações de Product. Seus construtores não possuem a textura e o ícone do tipo de maquiagem, pois estes são pré-definidos dentro da classe - cada tipo de maquiagem possui sua textura e seu ícone.

Na camada View ficam as entidades responsáveis em exibir os dados aos usuários. Ela é formada pelas activities e pelos recyclers.

Na camada Controller ficam as entidades responsáveis em lidar com os dados e informar a View o que ela precisa exibir.

Para a parte relacionada à realidade aumentada, foi utilizada a biblioteca “ARCore Augmented Faces wrapper without Sceneform” . Ela é uma versão modificada do exemplo de Augmented Faces disponibilizado no github da ARCore SDK , que torna mais simples a adição de texturas e objetos a um rosto. Ele é formado pelos diretórios Helpers (classes de auxílio) e Rendering (classes responsáveis pela renderização dos objetos RA e background) e pelos arquivos:

- AugmentedFaceFragment: classe responsável pela renderização e rastreamento das Augmented Faces.
- AugmentedFaceListener: interface responsável pelo callback de adicionar e atualizar as Augmented Faces.
- AugmentedFaceNode: classe responsável pela renderização do rosto incluindo as texturas e os objetos 3D.
- AugmentedFaceRenderer: classe responsável por renderizar a textura.
- FaceRegion: classe responsável por renderizar os objetos 3D de uma parte do rosto.

O uso da biblioteca é simples, basta adicionar o AugmentedFaceFragment ao layout e implementar a interface AugmentedFaceListener na activity de visualização com RA, adicionando a textura referente à maquiagem e sua cor.

6. Telas do MaquiAR

A seguir temos as telas finais do aplicativo MaquiAR:

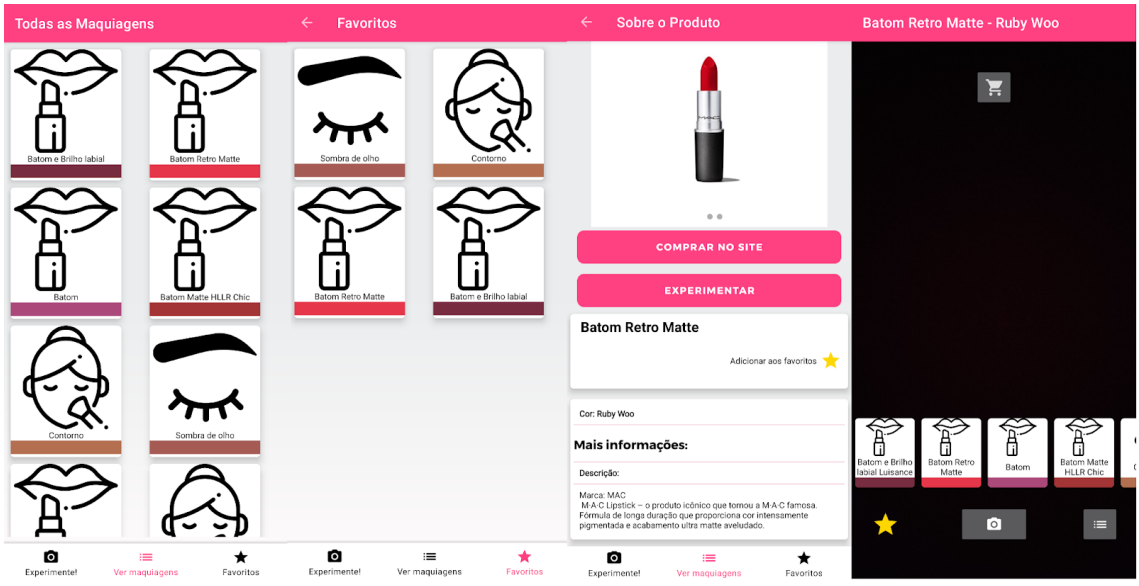


Figura 2. Tela principal, Favoritos, Detalhe do produto e Câmera

E abaixo temos exemplos dos filtros de maquiagem disponíveis no MaquiAR:

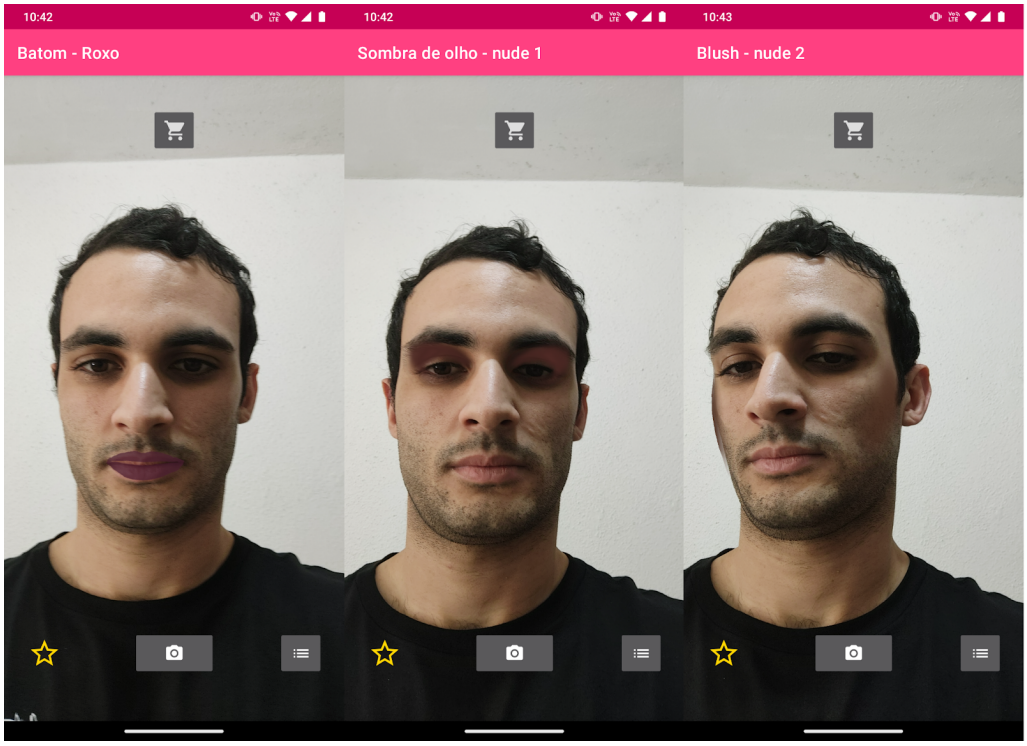


Figura 3. Filtros de batom, sombra de olho e blush do MaquiAR

7. Experimento: Teste com usuários

Foram convidados 10 estudantes da Universidade Federal de Santa Catarina, para participar da pesquisa. O experimento foi composto por quatro etapas: um questionário pré-uso do aplicativo, a utilização do aplicativo pelo participante, um questionário pós-uso do aplicativo e o questionário System Usability Scale (SUS).

O formulário pré-uso possui perguntas com o objetivo de avaliar a familiaridade do participante com compras de maquiagem online. Foram elaboradas perguntas diretas, com o objetivo de validar o problema.

A etapa de utilização do aplicativo pelo participante acontece sem interferência do condutor da pesquisa, ou seja, o participante não teve "dicas" de como utilizar o aplicativo. A única instrução que fora passada foi de que o participante deveria experimentar ao menos duas maquiagens diferentes. Essa decisão foi feita para que a usabilidade do aplicativo também seja testada.

O formulário pós-uso possui perguntas para avaliar a recepção do MaquiAR pelo usuário. Foram elaboradas 4 perguntas, utilizando a escala Likert de 5 pontos, e após cada uma havia um espaço livre para que o participante escrevesse o que quisesse relacionado a pergunta.

Por fim, com o objetivo de avaliar a usabilidade do MaquiAR, foi aplicado o SUS. O SUS é um questionário feito para medir a percepção de usabilidade. Ele foi criado por John Brooke em 1986 e é o questionário padrão para a indústria, sendo referenciado por mais de 600 publicações. (SAURO, 2016). Ele é constituído por 10 afirmações, cada uma com uma escala Likert de 5 pontos de concordância. É importante que a ordem original das perguntas seja mantida, para não afetar no resultado final.

7.1. Resultados

O questionário pré-uso revelou que os participantes do experimento foram jovens (entre 18 e 23 anos) e que costumam comprar maquiagem com certa frequência. Apenas um participante afirmou nunca ter comprado maquiagem pela internet. Revelou também que todos os participantes que compraram uma maquiagem pela primeira vez pela internet, sem ter experimentado anteriormente, tiveram algum tipo de arrependimento, sendo o principal motivo para tal arrependimento a cor do produto não ser a ideal para a pessoa. O questionário mostrou também que os participantes possuem algum receio de comprar maquiagem pela internet, mas que muitas vezes a comodidade do e-commerce supera esse receio.

Outro ponto interessante que o questionário pré-uso revelou foi que nenhum participante afirmou conhecer aplicativos de try-on de maquiagem. Num primeiro olhar, esse resultado é surpreendente, mas quando lembramos dos aplicativos do mercado - analisados na seção 3 deste trabalho - percebemos que os principais aplicativos de comércio eletrônico de maquiagens não possuem try-on, e que os aplicativos de try-on

de maquiagem não são muito utilizados para esse fim no Brasil, seja por não possuir produtos e lojas brasileiras, ou por ser mais usado como um aplicativo de edição de fotos. Um fato que podemos destacar é o desconhecimento, por parte dos participantes, de que o Instagram possui essa funcionalidade.

O questionário pós-uso revelou que os participantes consideraram as maquiagens apresentadas no MaquiAR como não muito realistas (média de 2,5 na escala de 5 pontos). Os participantes, porém, afirmaram que se sentiam mais confiantes para realizar uma compra online após utilizar o MaquiAR. Afirmaram também que a experiência utilizando o MaquiAR foi positiva e que utilizariam aplicativos de try-on de maquiagem sempre antes de realizar uma compra de maquiagem online.

A média do SUS score do MaquiAR para os 10 participantes foi de 88,5. Isso posiciona a usabilidade do MaquiAR acima da média da usabilidade de aplicativos móveis.

8. Conclusões e Trabalhos Futuros

Ao analisarmos os resultados obtidos pelo experimento, podemos tirar algumas conclusões. A recepção do uso da realidade aumentada na exibição de maquiagens foi muito positiva, tendo os participantes do experimento respondido que usariam aplicativos assim toda vez que fariam uma compra de maquiagem online. Outro ponto de destaque é que isso se deu mesmo com a avaliação de que as maquiagens apresentadas no MaquiAR não são muito realistas. Os motivos para tal avaliação não foram profundamente estudados, porém é importante salientar que os participantes utilizaram um dispositivo que não é o celular pessoal deles então a qualidade da câmera, diferente da que estão acostumados, pode ter afetado nas suas avaliações. Usuários afirmaram que apesar delas não terem algumas características da maquiagem real, as cores elas possuíam um grau de fidelidade que lhes trazia segurança para escolher a cor certa de um determinado produto.

Dessa forma, é esperado que, nos próximos anos, a realidade aumentada esteja cada vez mais presente com esse propósito, de apresentar produtos do e-commerce ao usuário, especialmente pois a tecnologia está se desenvolvendo e se tornando cada vez mais realista, o que trará ainda mais confiança ao comprador.

Obviamente, sempre há pontos de melhoria, em qualquer trabalho. Principalmente devido a restrição de tempo, algumas funcionalidades não foram implementadas. Segue sugestões de melhorias e de trabalhos futuros:

Sugestões de melhorias:

- Transformar as activities de listar produtos, PDP e favoritos em fragments, o que tornará a navegação no aplicativo mais fluida e melhorará a experiência do usuário.
- Tornar os botões da tela de try-on visualmente mais agradáveis.
- Filtrar os produtos, por categoria, cor, etc.
- Indicativo de qual produto está sendo experimentado na lista de produtos na tela de try-on.

Sugestões para trabalhos futuros:

- Implementar os requisitos funcionais desejáveis.
- Implementar servidor backend.
- Realizar o experimento para um grupo maior e mais heterogêneo
- Estudar os motivos dos participantes terem avaliado os filtros como não muito realistas
- Implementar sistema que utilize inteligência artificial para recomendar maquiagens ao usuário.

Referencias

BLOOMENTHAL, Andrew (2021) Electronic Commerce (Ecommerce). Investopedia. Disponível em: https://www.investopedia.com/terms/e/ecommerce.asp.

GOOGLE (2022) Introdução de rostos aumentados. Disponível em: https://developers.google.com/ar/develop/augmented-faces.

KIM, Jiyeon; FORSYTHE, Sandra (2008) Adoption of Virtual Try-on technology for online apparel shopping. Journal Of Interactive Marketing, [S.L.], v. 22, n. 2, p. 45-59. Elsevier BV. http://dx.doi.org/10.1002/dir.20113.

LU, Yuzhu; SMITH, Shana (2007) Augmented Reality E-Commerce Assistant System: trying while shopping. Lecture Notes In Computer Science, [S.L.], p. 643-652. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-540-73107-8_72.

SAMPAIO, Daniel (2019) O que é E-commerce? Tudo o que você precisa saber para ter uma loja virtual de sucesso! Rocketcontent. Disponível em: https://rockcontent.com/br/blog/e-commerce-guia/.

SAURO, Jeff (2016) Measuring Usability With The System Usability Scale (SUS). Disponível em: http://www.userfocus.co.uk/articles/measuring-usability-with-the-SUS.html.

SILVA, S.; ABREU, A.; TEIXEIRA, A. P. (2021) Influência da Realidade Aumentada na Decisão de Compra dos Consumidores. Research Bulletin (Cadernos de Investigação) of the Master in E-Business, [S. l.], v. 1, n. 1, 2021. Disponível em: https://www.iscap.pt/ebusiness-rj/index.php/mne-rj/article/view/58.

YIM, Mark Yi-Cheon; CHU, Shu-Chuan; SAUER, Paul L. (2017) Is Augmented Reality Technology an Effective Tool for E-commerce? An Interactivity and Vividness Perspective. Journal Of Interactive Marketing, [S.L.], v. 39, p. 89-103, Elsevier BV. http://dx.doi.org/10.1016/j.intmar.2017.04.001.