Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Ricardo Luna da Silva

Detecção de Primitivas Usando Soluções de Realidade Aumentada para Dispositivos Móveis

Recife

2020

Ricardo Luna da Silva

Detecção de Primitivas Usando Soluções de Realidade Aumentada para Dispositivos Móveis

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Mídia e Interação

Orientadora: Veronica Teichrieb

Coorientadores:
João Paulo Silva do Monte Lima
Rafael Alves Roberto

Recife

2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586d Silva, Ricardo Luna da
Detecção de primitivas usando soluções de realidade aumentada para dispositivos móveis / Ricardo Luna da Silva. – 2020.
69 f.: il., fig., tab.

Orientadora: Veronica Teichrieb.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.

Inclui referências.

1. Mídia e interação. 2. Dispositivos móveis. I. Teichrieb, Veronica (orientadora). II. Título.

006.7 CDD (23. ed.) UFPE - CCEN 2022-122

Ricardo Luna da Silva

“Detecção de Primitivas Usando Soluções de Realidade Aumentada para Dispositivos Móveis”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 29/10/2020.

_____

Orientadora: Profa. Dra. Veronica Teichrieb

BANCA EXAMINADORA

_____

Prof. Dr. Hansenclever de França Bassani
Centro de Informática/ UFPE

_____

Prof. Dr. Bruno José Torres Fernandes
Escola Politécnica de Pernambuco / UPE

_____

Prof. Dr. João Paulo Silva do Monte Lima
Departamento de Computação/UFRPE
(Co-orientador)

Dedico este trabalho à minha família por me ajudar durante toda minha trajetória e períodos de dificuldade.

AGRADECIMENTOS

Gostaria de agradecer a Deus pela ajuda durante esse período do mestrado e por tudo o que passei. Gostaria de agradecer também aos meus pais que me incentivam nesse caminho.

Gostaria de deixar um grande obrigado para toda equipe do Voxar Labs, que ofereceu todo o suporte necessário e onde fui acolhido. Meus mais sinceros agradecimentos para os meus orientadores, que com toda a paciência foram dando conselhos e ajudando nas decisões.

Esse período que passei no Centro de Informática na UFPE foi uma época que dificilmente vou esquecer, mas deixo os meus agradecimentos para todos que conheci durante esses 2 anos.

RESUMO

Aplicações de Realidade Aumentada geralmente utilizam uma técnica ou um conjunto de técnicas capaz de mapear, analisar, identificar e reconhecer o ambiente ao redor do usuário. A informação do espaço físico pode ser utilizada para adição de objetos virtuais, mapeamento ou localização no meio. Com um mapeamento do ambiente é possível sair de uma representação de mais baixo nível, como uma nuvem de pontos 3D, para uma representação de mais alto nível, como informações sobre formas ou objetos presentes no ambiente. Conhecer uma forma ou conjunto de primitivas geométricas pode ser útil em várias aplicações para compreensão da cena e engenharia reversa. As primitivas são sólidos geométricos (por exemplo, esferas, cilindros e planos) que compõem o objeto ou um conjunto de objetos. Com a evolução do *hardware* em dispositivos móveis, as possibilidades de uso e informações obtidas em tempo real estão aumentando, permitindo um uso em campos que antes não poderiam ser contemplados. Bibliotecas que utilizam esse tipo de *hardware* para o desenvolvimento de aplicações de Realidade Virtual e Realidade Aumentada estão sendo popularizadas, mas não sendo utilizada no contexto da detecção de primitivas em si. Esse trabalho utilizou nuvens de pontos geradas a partir de um dispositivo móvel utilizando o ARCore, biblioteca de Realidade Aumentada, como entrada para um algoritmo modificado a partir do *Efficient* RANSAC para efetuar a detecção de primitivas. Sendo esse tipo de abordagem não avaliada até o momento. Essa combinação demonstrou resultados significativos para detecção de primitivas como também evidenciou pontos de melhoria na etapa de mapeamento e na etapa de detecção.

Palavras-chaves: detecção de formas; detecção de primitivas; dispositivos móveis.

ABSTRACT

Augmented Reality applications usually apply a technique or a set of techniques able to map, analyze, identify and recognize the user environment. The information about physical space can be used to add virtual objects, mapping and perform tracking in the environment. With environment mapping, it is possible to change from a low-level representation like a 3D point cloud to a high-level representation, extracting information about objects or shapes presents in the scene. Knowledge about shapes or a set of geometric primitives can be useful for many applications to understand the scene and perform reverse engineering. The primitives are geometric solids (e.g. spheres, cylinders, planes) that compose the object or a group of objects. The evolution of mobile devices' hardware allows the possibility of usage and acquisition of information in real-time, allowing their use in fields that could not be applied previously. Libraries that use this kind of hardware for Augmented Reality and Virtual Reality development applications are being popularized but not being used in the context of detecting primitive. . This work applies point clouds generated from a mobile device that uses the ARcore, an Augmented Reality library, as input for a modified algorithm based on Efficient RANSAC that detects primitives. This type of approach has not been evaluated so far. This combination showed significant results for primitive detection as well as indicating points of improvement in the mapping and detection steps.

Keywords: shape detection; primitive detection; mobile devices.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN	*Convolutional Neural Network*
EKFs	Filtros de Kalman Estendidos
GS-IST	*Geometric and Statistical Incremental Semantic Tracking*
IMU	Unidade de Medida Inercial
IO	*Inertial Odometry*
Mems	Sistema Microeletromecânico
MLE	Estimativa por Máxima Verossimilhança
PnP	*Perspective-n-Point*
RA	Realidade Aumentada
RANSAC	*Random Sample Consensus*
RBPFs	Filtros de Partícula Rao-Blackwellized
RM	Realidade Mista
RV	Realidade Virtual
SLAM	Mapeamento e Localização Simultânea

LISTA DE SÍMBOLOS

$\mathcal{P}$	Nuvem de pontos
n_i	Normal do ponto i
Ψ	Conjunto de primitivas
$\mathcal{R}$	Conjunto de pontos restantes
τ	Tamanho de forma mínimo

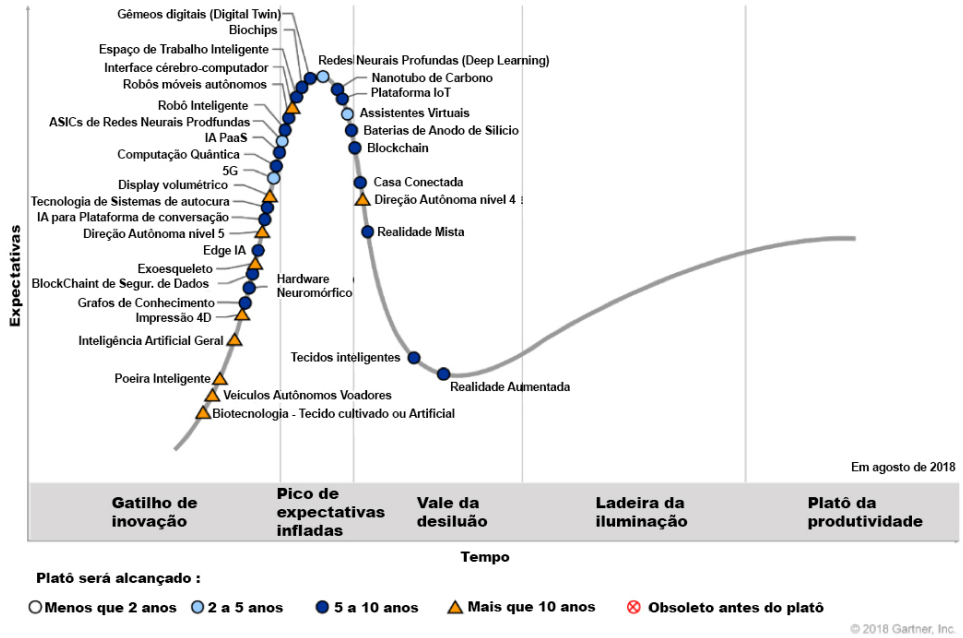
SUMÁRIO

1 INTRODUÇÃO

A empresa Gartner apresenta anualmente um estudo sobre tecnologias promissoras e seu amadurecimento que é representado através de um gráfico chamado *Gartner Hype Cycle* (GARTNER, 2018). Esse gráfico tenta apresentar de forma simples e clara uma representação da maturidade das tecnologias que estão sendo adotadas. A partir dele é possível ver a relevância de uma tecnologia e iniciar uma reflexão sobre quais são as tecnologias que estão surgindo, estão em desenvolvimento e já estão consolidadas.

Esse gráfico, apresentado na Figura 1, é dividido tentando apresentar os estágios de uma tecnologia: surgimento, quando ela tem suas expectativas infladas; vale da desilusão, quando o interesse pela tecnologia diminui e ela não cumpre o esperado; e platô da produtividade, que é quando foi encontrada uma clara aplicabilidade e relevância no mercado para a tecnologia em questão.

Figura 1 – Gráfico Hype Cycle de tecnologias emergentes de 2018, o texto na imagem foi traduzido.



Fonte: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/.

Segundo a Gartner, é possível observar o tempo estimado para uma tecnologia alcançar o platô da produtividade. Nessa figura, duas das tecnologias (Realidade Mista - RM e Realidade Aumentada - RA) já estão passando pelo vale da desilusão e o tempo estimado para que elas possam ser aplicadas fortemente no mercado é de 5 até 10 anos. Já o Hype

Cycle de 2020[1] não apresenta RM e RA, que acabaram saindo dessa lista já em 2019, mas indicando que essas tecnologias podem ter alcançado seu platô antes do período estimado anteriormente em 2018. Já na Figura 2 são apresentadas as principais tendências tecnológicas para 2019, e entre elas figura a tendência de Experiência Imersiva. Esse tipo de tecnologia tem uma forte base em RA, RM e Realidade Virtual (RV) pela sua capacidade de mudar como o usuário percebe o ambiente.

Figura 2 – Principais tendências tecnológicas estratégicas para 2019 segundo a Gartner.

Fonte:
https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/.

Foi utilizando esse contexto, com a RA consolidada e também como tendência, como entrada que nossa atenção é voltada em uma tecnologia que ganhou muito destaque com o lançamento do jogo Pokémon GO (NIANTIC INC., 2019). Esse jogo foi lançado em 2016

[1] https://www.gartner.com/smarterwithgartner/5-trends-drive-the-gartner-hype-cycle-for-emerging-technologies-2020/

pela Niantic, Inc. e Nintendo. O sucesso do jogo se deu pelo uso de RA em dispositivos móveis, sendo compatível com muitos dispositivos no mercado, e da localização do jogador mostrando a importância dessa área. O jogo ficou famoso mundialmente pouco tempo depois de seu lançamento. Na tela do seu *smartphone*, o jogador pode ver a adição de um Pokémon ao mundo real, como mostrado na Figura 3.

Figura 3 – Pokémon GO.

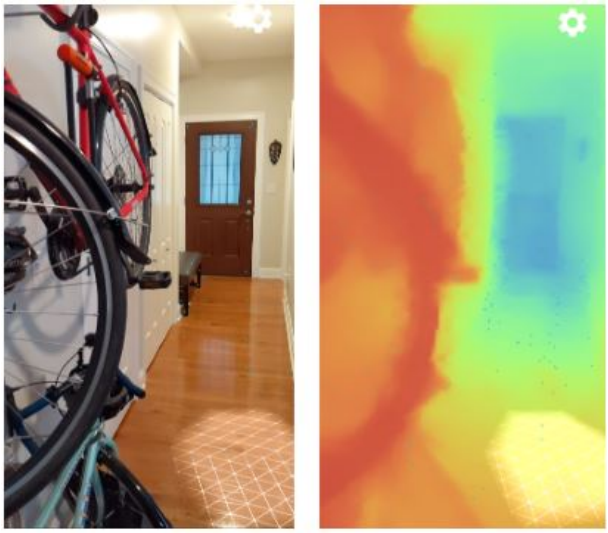
Fonte: https://mundoconectado.com.br/noticias/v/2756/pokemon-go-recebera-modo-co-op-em-breve-anuncia-niantic.

A evolução dos dispositivos móveis e sua popularização fez que eles se tornassem um item essencial para a maioria das pessoas. Os dispositivos ficaram mais leves, compactos e são geralmente utilizados em quase todo os ambientes. Esses foram alguns dos motivos para que tais dispositivos fossem escolhidos e aplicados no contexto de detecção de formas, pois possui todo o *hardware* necessário em mãos tornando seu uso mais natural. No ano de 2020, uma biblioteca de RA chamada ARCore[2], desenvolvida pelo Google, adicionou uma nova funcionalidade nomeada como *Depth* API, que fornece um mapa de profundidade de baixa resolução do ambiente observado, como pode ser visto na Figura 4. Isso possibilita a utilização de dispositivos móveis em mais aplicações, pois com o uso dessa API se tornou possível obter uma nuvem de pontos densa a partir de um dispositivo móvel, o que antes era feito mais comumente com o Microsoft Kinect e sensores similares.

RA possui uma definição que é bem aceita no meio acadêmico. Segundo Azuma (1997), para ser considerada RA, uma dada tecnologia deve apresentar três requisitos: combinar conteúdo real e virtual, a aplicação deve ser interativa em tempo real e realizar o registro em 3D. Para conseguir cumprir corretamente os três requisitos, a RA usa a infraestrutura de outras áreas, tais como Visão Computacional, Computação Gráfica e Interação, tendo

[2] https://developers.google.com/ar

Figura 4 – Imagem de entrada (esquerda) e mapa de profundidade gerado pelo ARCore (direita).



Fonte: https://developers.google.com/ar/develop/unity/depth/overview.

como seu principal foco melhorar a experiência dos usuários (ZHOU; DUH; BILLINGHURST, 2008).

Para combinar conteúdo real e virtual, é necessário aplicar alguma técnica que forneça informações sobre o ambiente ou um conjunto delas: luminosidade, localização espacial ou forma dos objetos contidos no ambiente, entre outras. Para um posicionamento correto de informações virtuais em relação à realidade, chamado de registro, percebe-se que é requerido um conhecimento do ambiente real e essa informação deve ser obtida de forma rápida, assim visando um tempo de resposta aceitável para o usuário da aplicação. O rastreamento de objetos 3D, forma de obter informações sobre a posição e orientação de objetos ao longo do tempo, pode ser resolvido utilizando estratégias *bottom-up* ou *top-down*. Na abordagem *bottom-up*, são inicialmente extraídas características a partir da imagem para em seguida determinar a pose do objeto. Já utilizando a *top-down*, inicialmente é criada a hipótese de pose com informações presentes na imagem, então essas hipóteses são avaliadas visando encontrar a que mais se aproxima da pose real do objeto rastreado.

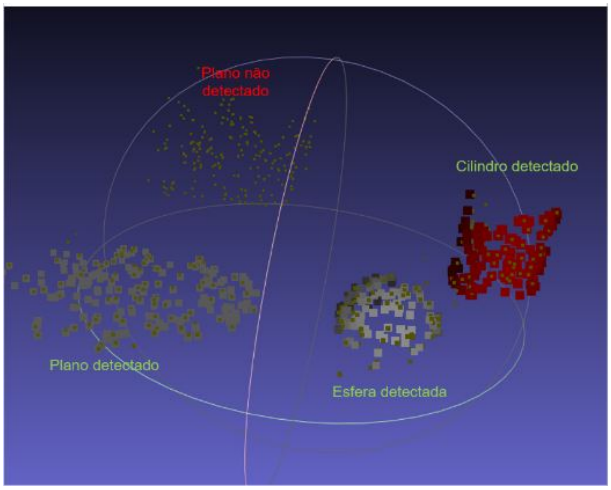
Alguns dispositivos, como por exemplo o Microsoft Kinect, foram criados com o propósito de obter informações 3D do ambiente, sendo capazes de realizar a aquisição de tais informações em tempo real. Os dados sobre o ambiente geralmente são representados como uma nuvem de pontos. Esse tipo de representação se mostra suficiente para o reconhecimento de formas.

Depois do sucesso de aplicações utilizando RA (como por exemplo jogos e aplicações

voltadas para vendas/indústria), foi observado um certo interesse no desenvolvimento de bibliotecas para criação de forma rápida e simples de aplicações de RA. Através dessas bibliotecas ocorre também a evolução contínua de técnicas e novas funcionalidades que agregam mais informações para aplicações cada vez mais realistas.

A modelagem semântica se torna poderosa por conseguir identificar formas e outros padrões para criar um melhor entendimento do mundo ao redor do usuário. Desta forma se obtém uma maior compreensão da cena e se torna possível aplicar engenharia reversa para entender sobre informações de oclusão ou até colisão física com relação aos objetos observados. Quando se pode estimar primitivas geométricas a partir de nuvens de pontos, como visto na Figura 5, se torna praticável a criação de uma representação da cena de forma mais simples e usar essa informação como referência para aplicações como navegação autônoma, reconhecimento de objetos e sistemas de RA.

Figura 5 – Primitivas detectadas a partir de uma nuvem de pontos.



Fonte: elaborado pelo autor .

Como apresentado em Kaiser, Zepeda e Bouberkeur (2019), a detecção de primitivas apresenta alguns problemas e desafios. As câmeras de profundidade são ótimas fontes para capturas de dados 3D, mas geralmente possuem um preço um pouco mais elevado. Algumas abordagens podem apresentar problemas com desempenho quando avaliadas com restrições de funcionamento em tempo real. Outro fator (KAISER; ZEPEDA; BOUBEKEUR, 2019) é a interpretação dos dados que vão além de apresentar uma lista com o conjunto de formas detectadas. Informações referentes à posição e à orientação do objeto podem ser úteis dependendo da aplicação.

Em Roberto et al. (2019) são apresentados casos em que a detecção de primitivas não

funcionou em tempo real em dispositivos móveis. Um dos fatores causadores é o tamanho da nuvem de pontos avaliada. No mesmo trabalho é sugerido o uso de bibliotecas de RA como sistema de SLAM devido ao seu funcionamento em tempo real, mas nele não é apresentada nenhuma avaliação sobre elas. Começar por essa avaliação pode mostrar se essa abordagem vale a pena e quais suas limitações.

1.1 DEFINIÇÃO DO PROBLEMA E OBJETIVOS

Visando resolver questões relacionadas com a problemática citada anteriormente, uma abordagem interessante é utilizar dispositivos móveis. Tal fato vem da premissa de que não seria necessário o uso de *hardware* adicional como sensores de profundidade e de que as informações obtidas a partir de seus sensores podem ser utilizadas em técnicas de SLAM para obter um resultado melhor. Outro fator importante para essa decisão é que já foram criadas bibliotecas para dispositivos móveis que tem como principal foco o desenvolvimento de aplicações de RA, implementando os principais algoritmos e explorando de forma eficiente o *hardware* existente no dispositivo para obter as informações necessárias para a aplicação. A principal questão relacionada ao tema é: "Podemos utilizar dispositivos móveis e bibliotecas de RA para detecção de primitivas?".

Dado o problema de detectar primitivas geométricas sem utilização de *hardware* adicional além de *smartphones* ou *tablets*, esse trabalho procura buscar soluções para algumas perguntas de pesquisa que são:

- Q1: É possível detectar primitivas utilizando apenas informações advindas de um dispositivo móvel?
- Q2: Quais as limitações do uso de dispositivos móveis para esse contexto?
- Q3: Quais as melhorias que podem ser realizadas para uma evolução nos resultados já existentes?

Os objetivos específicos desse trabalho são:

- Buscar um entendimento sobre técnicas relacionadas a detecção de primitivas;
- Entender quais dessas técnicas podem ser utilizadas no contexto dessa pesquisa;
- Observar a partir dos experimentos quais são suas limitações e quais melhorias podem ser realizadas nesse contexto.

1.2 ESTRUTURA DO DOCUMENTO

Este trabalho apresenta um total de 5 capítulos:

- O Capítulo 2 apresenta uma contextualização sobre a área de pesquisa e os trabalhos relacionados ao tema da dissertação.
- No Capítulo 3 são apresentadas as abordagens desenvolvidas para detecção de primitivas em dispositivos moveis.
- O Capítulo 4 detalha os resultados obtidos e as avaliações realizadas.
- O Capítulo 5 apresenta uma conclusão sobre o que foi abordado no trabalho, descrevendo também quais as são as contribuições e os trabalhos futuros.

2 CONTEXTUALIZAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Como já apresentado anteriormente, uma aplicação para ser considerada RA deve cumprir três requisitos, mas para cumprir esses requisitos devemos obter algumas informações espaciais 3D do ambiente para assim conseguir adicionar um objeto virtual ao mundo real de forma que o objeto pareça fazer parte do mundo físico.

O registro baseado em visão, como citado em Uenohara e Kanade (1995), é um processo que utiliza características da imagem para realizar a correspondência dessas características com informações de um modelo e assim a pose do objeto será estimada e o objeto virtual poderá ser adicionado na localização correta.

Para realizar o rastreamento, como citado em Azuma (1997), um conjunto de técnicas podem ser aplicadas e algumas delas são: rastreamento magnético, rastreamento baseado em visão (utilizando imagens como entrada), rastreamento utilizando Unidade de Medida Inercial (*Inertial Measurement Unit* - IMU) e rastreamento híbrido. O uso de cada técnica pode depender do contexto do problema ou do *hardware* disponível.

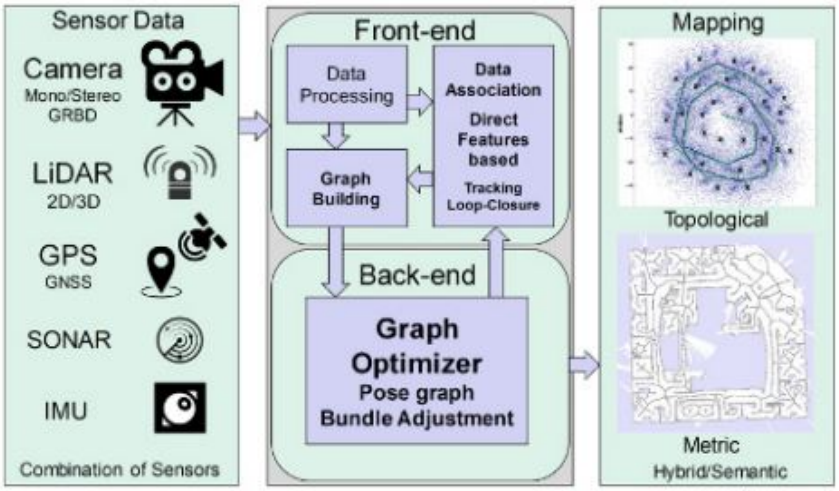
No contexto da robótica, é utilizada uma abordagem chamada de Mapeamento e Localização Simultâneos (*Simultaneous Localization and Mapping* - SLAM), que pode ser aplicada para um robô autônomo criar uma representação do ambiente em tempo real enquanto calcula e obtém informação sobre sua posição e orientação.

2.1 MAPEAMENTO E LOCALIZAÇÃO SIMULTÂNEOS

Tipos clássicos de SLAM, segundo Caneda et al. (2016), possuem abordagens baseadas em Filtros de Kalman Estendidos (*Extended Kalman Filters* - EKFs), Filtros de Partícula Rao-Blackwellized (*Rao-Blackwellized Particle Filters* - RBPFs) e Estimativa por Máxima Verossimilhança (*Maximum Likelihood Estimation* - MLE). Em Taketomi, Uchiyama e Ikeda (2017) também são apresentados os tipos de SLAM que são considerados atualmente como o estado da arte: MonoSLAM (DAVISON et al., 2007), PTAM (KLEIN; MURRAY, 2007), DTAM (NEWCOMBE; LOVEGROVE; DAVISON, 2011), LSD-SLAM (ENGEL; SCHÖPS; CREMERS, 2014) e SLAM++ (SALAS-MORENO et al., 2013). Como este trabalho tem como objetivo o uso de um dispositivo móvel como *hardware* para uso, devemos voltar nosso foco para dois tipos de SLAM: SLAM Visual (*Visual SLAM* - VLSAM) e SLAM Visual-Inercial

(*Visual-Inertial SLAM* - VISLAM). Esses são dois tipos que podem ser utilizados no nosso contexto, pois o *hardware* de captura em um dispositivo móvel seria a câmera RGB e também os sensores inerciais (acelerômetro, giroscópio e magnetômetro) do dispositivo. A Figura 6 apresenta a arquitetura de um sistema de SLAM.

Figura 6 – Arquitetura de uma sistema de SLAM.



Fonte: Sualeh e Kim (2019).

Odometria Inercial (*Inertial Odometry* - IO), como citado em Mohamed et al. (2019), é um método que atualiza a localização de um dispositivo usando apenas os dados fornecidos por uma IMU, que atualiza sua posição, orientação, altitude e velocidade linear. Ainda citado por Mohamed et al. (2019), essa IMU é um Sistema Microeletromecânico (*Microelectromechanical System* - MEMS) que tem como construção o uso de um acelerômetro e giroscópio. O grande problema desse tipo de abordagem é que a mesma pode sofrer de erros oriundos de medições imprecisas ou inadequadas do giroscópio e acelerômetro.

Segundo Taketomi, Uchiyama e Ikeda (2017), os módulos principais em um sistema que utiliza a abordagem do VSLAM são: inicialização, rastreamento e mapeamento. Uma parte fundamental para o sistema é definir um sistema de coordenadas inicial e assim, a partir dessa etapa, realizar a estimativa de pose da câmera. A pose consiste na localização da câmera no espaço 3D, que é representada por uma matriz que contém rotação e translação. Na etapa de rastreamento o mapa global que está sendo construído é rastreado a partir da imagem atual para identificar a nova localização da câmera nesse sistema de coordenadas. O mapa armazena pontos 3D que foram obtidos anteriormente. Esses pontos são características que apareceram na imagem e, à medida que novas imagens são obtidas, o mapa vai sendo expandido. O rastreamento, a partir das novas imagens obtidas,

tenta buscar correspondências entre características 2D identificadas nas imagens obtidas e pontos 3D do mapa, que são relacionadas através da correspondência de características ou rastreamento delas. Já a pose da câmera pode ser obtida utilizando correspondências resolvendo um problema de *Perspective-n-Point* (PnP), como é citado em Taketomi, Uchiyama e Ikeda (2017).

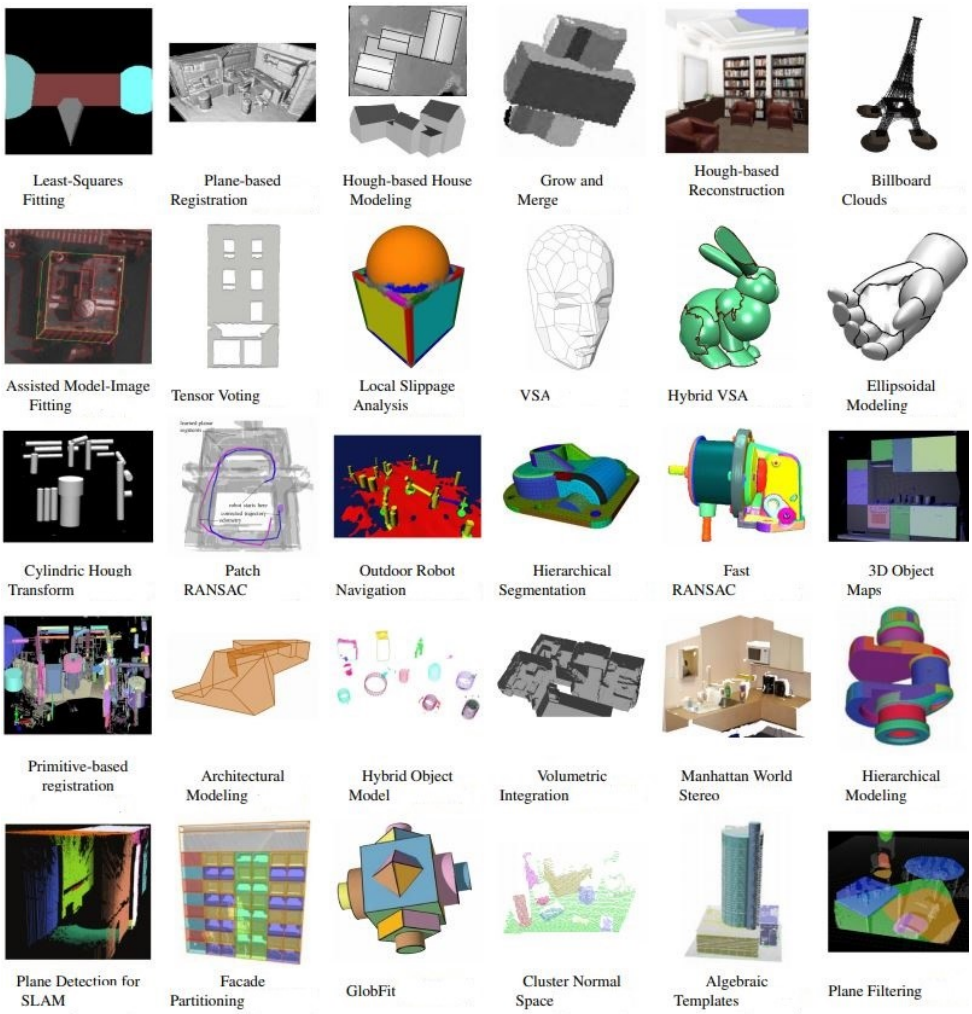
Ainda segundo Taketomi, Uchiyama e Ikeda (2017), outros dois módulos que ainda são encontrados em sistemas que utilizam o VSLAM são: relocalização e otimização de mapa global. A relocalização é um módulo de essencial importância quando o sistema falha ou de alguma forma o rastreamento é perdido, mas para recuperar tal informação a pose da câmera deve ser recalculada com relação ao mapa novamente. Tal módulo tem sua importância pelo fato de que se o sistema perde sua localização poderia não funcionar mais corretamente. Já o modulo de otimização global é utilizado pois o mapa pode gerar acúmulos de erros de estimativa, sendo função desse modulo refinar o mapa levando em consideração a consistência de toda a informação obtida até o momento. O erro acumulado da pose pode ser obtido quando um fechamento de laço (*loop closure*) é identificado, que acontece quando a câmera observa uma região que já foi mapeada novamente, e durante esse processo é realizada uma etapa similar a utilizada durante a relocalização.

Em Jinyu et al. (2019) os sistemas de SLAM são classificados como sendo sistemas por filtragem ou otimização. Os sistemas baseados em filtragem utilizam filtros para encontrar a pose da câmera, como por exemplo o MonoSLAM (DAVISON et al., 2007) que utiliza o filtro de Kalman. Os sistemas baseados em otimização utilizam informações obtidas, como por exemplo o quadro-chave, para realizar otimizações e atualizar a pose câmera minimizando o erro de reprojeção, como no caso do PTAM (KLEIN; MURRAY, 2007). Dentre os que utilizam filtragem temos: MonoSLAM (DAVISON et al., 2007), MSCKF (MOURIKIS; ROUMELIOTIS, 2007) e MSCKF 2.0 (LI; MOURIKIS, 2012). Já os sistemas baseados em otimização são: CHOLMOD (CHEN et al., 2008), COLAMD (DAVIS et al., 2004), ISAM (KAESS; RANGANATHAN; DELLAERT, 2008), ISAM 2.0 (KAESS et al., 2012), SLAM++ (ILA et al., 2017), PTAM (KLEIN; MURRAY, 2007), OKVIS (LEUTENEGGER et al., 2015), VINS-Mono (QIN; LI; SHEN, 2018), ORB-SLAM (MUR-ARTAL; TARDÓS, 2017b) e ORB-SLAM2 (MUR-ARTAL; TARDÓS, 2017a).

2.2 DETECÇÃO DE PRIMITIVAS

São vários os cenários onde a detecção de primitivas pode ser aplicada, segundo o trabalho de Kaiser, Zepeda e Boubekeur (2019). Alguns desses cenários são: robótica, modelagem 3D, processamento de forma, renderização, interação, animação e arquitetura. Esses métodos de detecção de primitivas podem ser agrupados (KAISER; ZEPEDA; BOUBEKEUR, 2019), da seguinte forma: estocásticos, de espaço de parâmetros ou por técnicas de agrupamento (*clustering*). A Figura 7 apresenta alguns dos algoritmos citados em Kaiser, Zepeda e Boubekeur (2019).

Figura 7 – Algoritmos de detecção de primitivas.

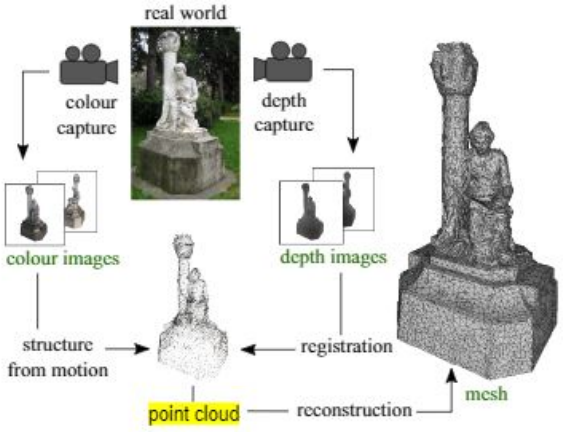


Fonte: Kaiser, Zepeda e Boubekeur (2019).

Cada tipo de cenário pode oferecer um tipo diferente de dado. O tipo de dado 3D pode ter diferentes representações, favorecendo assim determinadas técnicas de detecção, como citado em Kaiser, Zepeda e Boubekeur (2019). Para a maioria dos algoritmos utilizados para detecção de primitivas, são utilizados como entrada os seguintes dados: imagem,

nuvem de pontos e malha. Como apresentando em Kaiser, Zepeda e Boubekeur (2019), as imagens utilizadas podem ser RGB-D ou RGB. O D do RGB-D representa a profundidade (*Depth*), que pode ser obtida a partir de um tipo diferente de câmera. Já o RGB é um sistema de cores aditivo consistindo na combinação dos canais Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*), representação padrão obtida por câmeras, embora esse formato necessite de algum pós-processamento para obtenção dos dados necessários. Na Figura 8 é apresentado o exemplo de um *pipeline* de aquisição de um modelo 3D. As etapas que estão com a legenda verde indicam quais os possíveis estágios para utilização como entrada para a detecção de primitivas.

Figura 8 – *Pipeline* de aquisição 3D para a estátua Arenes de Lutece, Paris, França.



Fonte: Kaiser, Zepeda e Boubekeur (2019).

Ainda segundo Kaiser, Zepeda e Boubekeur (2019), outro formato utilizado é a nuvem de pontos, que é o formato padrão para esse tipo de aplicação, onde cada valor 3D (X, Y, Z) é a representação de um ponto no espaço real. Esse tipo de formato é geralmente representado por uma lista de posições 3D, podendo também conter informações opcionais como a cor que foi obtida a partir do ponto onde essa coordenada foi extraída.

A detecção de primitivas utilizando o método estocástico *Random Sample Consensus* (RANSAC) (FISCHLER; BOLLES, 1981) foi realizada em Schnabel, Wahl e Klein (2007). Nesse trabalho as primitivas detectadas foram: planos, esferas, cilindros, cones e toros. Esse trabalho utilizou como entrada uma nuvem de pontos orientada e não organizada.

Como citado por Kaiser, Zepeda e Boubekeur (2019), outro conjunto de algoritmos utiliza agrupamento, como em Lukács, Martin e Marshal (1998) e Oesau, Lafarge e Alliez (2016). Com esse tipo de abordagem, cada ponto da representação do objeto é considerado um indivíduo, e se inicializa utilizando um conjunto de sementes sendo escolhidas de

forma aleatória. Através de uma função de similaridade, esses indivíduos são agrupados e rotulados como membros de um determinado grupo. Uma técnica similar também foi utilizada para dividir os dados em regiões menores e posteriormente agrupar essas regiões que compõe o mesmo objeto. Para tal abordagem outra técnica deve ser aplicada. Já foi observado nesse contexto o uso do ajuste de retângulo (*rectangle fitting*) (MATTAUSCH et al., 2014; OESAU; LAFARGE; ALLIEZ, 2016), usando interpolação linear (THIERY; GUY; BOUBEKEUR, 2013) e fusão baseada em vizinhança (ZHOU et al., 2015).

Na área de aprendizagem de máquina, Li e Feng (2018) utilizaram uma combinação diferente das apresentadas anteriormente. Como entrada foi utilizada uma nuvem de pontos e para o processamento dos dados foi utilizada uma Rede Neural Convolucional (*Convolutional Neural Network* - CNN). Nos últimos anos o uso de CNN foi bastante popularizado. Com o avanço e o barateamento das placas gráficas, o uso dessa técnica passou a ser mais explorado e demonstrou resolver uma grande gama de problemas.

O trabalho em Roberto et al. (2017) apresenta também a possibilidade de identificar primitivas geométricas a partir de nuvens de pontos e refina de forma incremental seu resultado. Aqui a informação temporal é levada em conta, pois inicialmente a nuvem de pontos gerada não possui a informação sobre todo o formato do objeto, mas essa informação pode ser fundida e assim gerar a forma real do objeto observado.

Já em Roberto et al. (2019), um trabalho posterior, apresenta uma abordagem para detecção de primitivas utilizando análises geométricas e estatísticas em nuvem de pontos esparsas em dispositivos móveis. Também foi adicionada uma etapa de fusão de primitivas utilizando critérios de similaridade, mostrando ser superior se comparada com a abordagem que utiliza puramente o *Efficient RANSAC*, que apenas identifica primitivas presentes na nuvem de pontos apresentada. O artigo apresenta o *Geometric and Statistical Incremental Semantic Tracking* (GS-IST) que utiliza como entrada as nuvens de pontos geradas por um sistema SLAM e é capaz de criar análises utilizando a informação temporal para filtrar e classificar de forma mais confiável as primitivas geométricas.

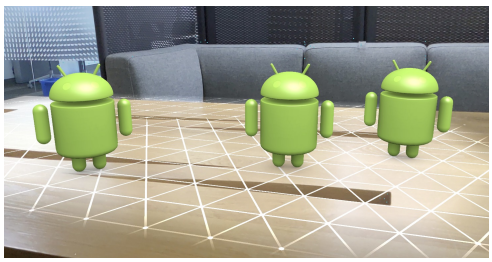
2.3 BIBLIOTECAS DE REALIDADE AUMENTADA PARA DISPOSITIVOS MÓVEIS

Segundo Jiny et al. (2019), as tecnologias de RA estão se desenvolvendo com uma grande velocidade e evoluindo rapidamente. Grandes empresas como Google, Apple e Microsoft lançaram *softwares* para RA focados em dispositivos móveis. Como a maioria desses dis-

positivos apresenta câmera e IMU, como apontado em Jiny et al. (2019), esses dispositivos são ideais para o VISLAM. Duas bibliotecas que estão fazendo uso disso são o ARCore[1] da Google e o ARKit[2] da Apple.

O ARCore usa um conjunto de diferentes APIs para permitir que o dispositivo seja capaz de detectar o ambiente, entender o mundo e assim criar interações a partir disso. Algumas das funcionalidades oferecidas pelo ARCore são: rastreamento, compreensão do ambiente e estimativa de luz. O rastreamento permite que o telefone seja capaz de identificar movimentos e rastrear sua posição em relação ao mundo. O dispositivo também se torna capaz de identificar superfícies horizontais e verticais, como visto da Figura 9, obtendo uma compreensão maior sobre dimensões do mundo. O ARCore também possui a capacidade de estimar a luz para entender as condições de iluminação para cada tipo de ambiente e gerar nuvens de pontos esparsas.

Figura 9 – Detecção de planos do ARCore.



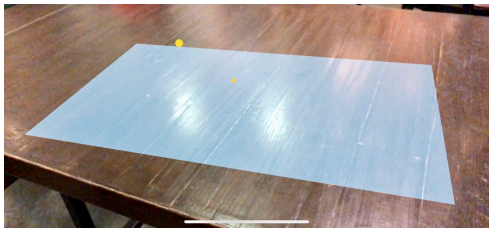
Fonte: https://developers.google.com/ar.

Já o ARKit apresenta um conjunto de funcionalidade similares ao ARCore, mas tendo como principal diferença a forma como algumas funcionalidades são implementadas. O ARKit funciona apenas em dispositivos da Apple e esse fator pode se tornar um problema devido ao preço de alguns deles. Por sua vez, o ARCore pode funcionar em uma lista maior de dispositivos, com sistema operacional Android, mas ainda possuindo um lista de dispositivos compatíveis[3] devido a restrições impostas pelo *hardware* dos mesmos. O ARKit também é capaz de detectar planos, como visto na Figura 10, gerar nuvens de pontos esparsas da cena e outros objetos 3D, porém esses objetos devem ser previamente escaneados e a representação criada do objeto só pode ser "vista" dentro da aplicação, não sendo possível exportar esse modelo.

O ARCore já foi explorado como parte de um sistema de navegação assistida em

[1] https://developers.google.com/ar
[2] https://developer.apple.com/augmented-reality/
[3] https://developers.google.com/ar/discover/supported-devices

Figura 10 – Detecção de planos do ARKit.



Fonte: https://www.appcoda.com/arkit-horizontal-plane/

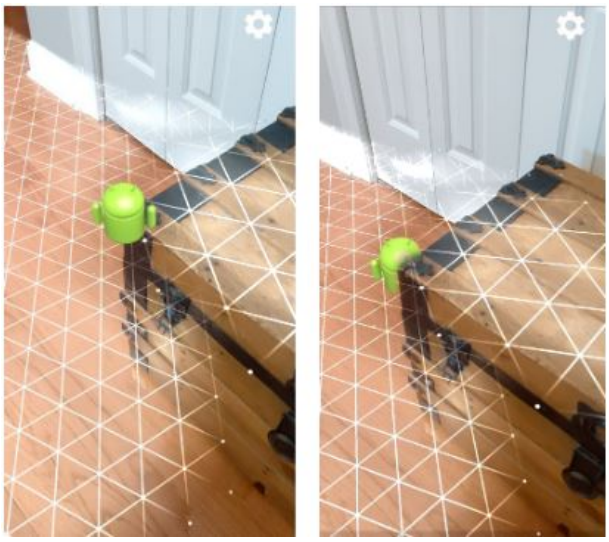
outro trabalho Zhang et al. (2019). Em Dilek e Erol (2018), o ARKit foi utilizado para desenvolvimento de um aplicativo para detectar a posição física e tal informação pode ser utilizada em aplicações de navegação.

O Wikitude[4] apresenta o rastreamento de objetos e cenas, uma imagem ou um conjunto, cilindro, conta com reconhecimento em nuvem, rastreamento de múltiplos objetos e gera a nuvem de pontos. Ele utiliza como suporte para algumas de suas funcionalidades o ARkit, ARcore e AR Foundation[5]. Mas diferentes dos outros dois, algumas das funcionalidades requerem aquisição da versão paga[6] e apresenta uma restrição de funcionalidades de acordo com o dispositivo[7] utilizado. Diferentes bibliotecas são citadas na literatura, tais como ARToolKit[8] e Vuforia[9], mas não possuem todas as funcionalidades apresentadas nas outras duas, ARCore e ARKit.

No início do ano de 2020, o ARCore divulgou algumas informações sobre seu novo trabalho recente e em junho do mesmo ano lançou a *Depth API*, que pode ser utilizada para trazer mais realismo às aplicações. As colisões, onde um objeto virtual colide com um objeto real, e as oclusões, que acontecem quando um objeto virtual é posicionado corretamente em frente ou atrás de um objeto real, passam a ser feitas de forma mais real, dado que agora são gerados mapas de profundidade da cena, mesmo que com uma resolução mais baixa que a imagem capturada. Na Figura 11 pode ser visto um caso de oclusão e na Figura 12 um caso de colisão.

[4] https://www.wikitude.com/
[5] https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html
[6] https://www.wikitude.com/store/
[7] https://www.wikitude.com/documentation/latest/android/supporteddevicesandroid.html
[8] http://www.hitl.washington.edu/artoolkit/
[9] https://developer.vuforia.com/

Figura 11 – Funcionamento da oclusão do ARCore antes (esquerda) e depois (direita) da *Depth* API



Fonte: https://developers.google.com/ar.

Figura 12 – Utilização do mapa de profundidade para gerar colisões mais realistas.

Fonte: elaborado pelo autor.

3 DETECÇÃO DE PRIMITIVAS EM DISPOSITIVOS MÓVEIS

Esse capitulo será divido em quatro seções para uma melhor explanação sobre cada um dos métodos utilizados. A primeira discute sobre as técnicas que foram avaliadas e os testes iniciais para a escolha e qual foi selecionada. A segunda seção fala sobre a método utilizado para o mapeamento dos objetos utilizando uma nuvem de pontos esparsa. A terceira seção fala sobre a nuvem de pontos densa gerada pela *Depth API*. Na quarta seção é detalhada a abordagem utilizada para detecção de primitivas, quais as etapas empregadas e quais os motivos do uso dessa técnica no atual contexto.

3.1 SELEÇÃO DA TÉCNICA PARA MAPEAMENTO

No início da pesquisa foi dado um grande foco na técnica de mapeamento que seria utilizada devido ao uso de dispositivos móveis. O uso de bibliotecas para desenvolvimento de aplicações de RA foi bastante motivado pelo desempenho oferecido durante os testes com aplicações nos dispositivos. Para a análise inicial, três bibliotecas foram selecionadas: ARKit, ARCore e Wikitude. O ARCore e Wikitude são bibliotecas que são compatíveis com dispositivos com Android e também podem ser utilizadas através do motor de jogos Unity[1]. O ARKit é a única biblioteca que funciona exclusivamente com dispositivos móveis da Apple. Para os experimentos com o ARCore foi utilizado o Samsung Galaxy S9. Já nos experimentos com a ARKit foi utilizado um IPad Pro modelo A1701. Para os experimentos com o Wikitude foi utilizado o Samsung Galaxy S9 e o Asus Zenfone 5. Cada uma das bibliotecas possui uma lista de dispositivos compatíveis e também apresenta algumas restrições com relação a plataforma de desenvolvimento, um exemplo é que o Wikitude pode ser utilizando no Zenfone 5 enquanto o ARCore não.

O ARCore apresenta algumas funcionalidades básicas como: rastreamento de movimento, compreensão do ambiente (detectando o tamanho e localização de superfícies) e estimação de luz. Essa biblioteca utiliza sensores inerciais e também o VSLAM para obter algumas dessa informações. As outras duas bibliotecas também apresentam características similares, com algumas diferenças na questão de desempenho. Para o experimento inicial foi avaliada a nuvem de pontos obtida a partir de cada uma das bibliotecas. Após o ma-

[1] https://unity.com/pt

peamento de um conjunto de objetos, a nuvem de pontos era armazenada no dispositivo e avaliada utilizando o MeshLab. Esse primeiro teste serviu apenas para visualizar como era o resultado obtido por cada uma das bibliotecas.

O ARCore apresenta um quarto e quinto valor para cada um dos pontos da sua nuvem. O quarto valor representa a confiança do ponto e pode ser utilizado como critério de escolha para quais os pontos que serão usados. E cada ponto é representado por um ID (quinto valor) que pode ser utilizado para identificar cada ponto durante uma sessão (execução). Para cada nova imagem é gerada uma nuvem de pontos para aquele quadro. Aqui não existe um acúmulo ou uma nuvem global disponibilizada pela biblioteca. Quando os pontos são exportados a cada imagem, ocorre acúmulo de erro evidenciado por um pequeno deslocamento entre nuvens de pontos subsequentes.

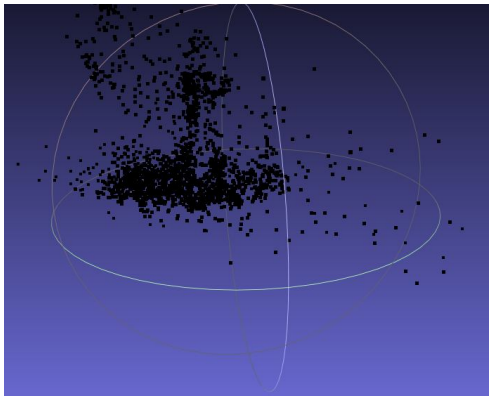
O ARKit apresenta uma forma para acessar a nuvem global, que é constituída por todos os pontos adicionados no mapa durante toda a execução da aplicação. Durante os experimentos iniciais, a nuvem final obtida possuía muito ruído e nem sempre era possível visualizar corretamente os objetos contidos no ambiente. Na Figura 13 é apresenta o objeto mapeado pelo ARKit e as Figuras 14 e 15 mostram a nuvem de pontos geradas. Essa nuvem de pontos contém uma grande quantidade de pontos fora da região do objeto ou dos planos exibidos.

Figura 13 – Objeto mapeado pelo ARKit.

Fonte: elaborado pelo autor.

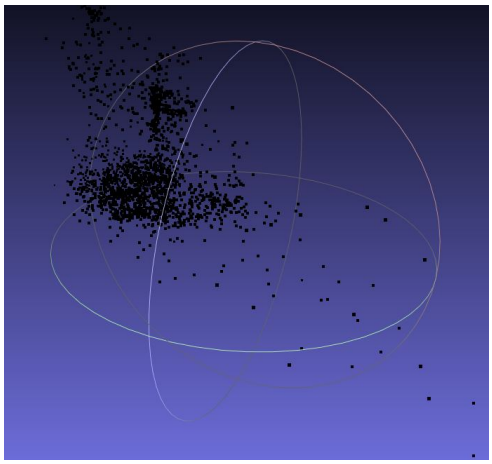
Já no caso do Wikitude, o mesmo apresentava um pequeno erro na coordenada y de cada ponto da nuvem, fazendo com que o objeto ficasse achatado por esse motivo. Ao observar o erro foi utilizado outro dispositivo (Zenfone 5) para os testes, mas o erro persistiu. A Figura 16 mostra o resultado obtido pelo Wikitude. A escala da coordenada

Figura 14 – Vista 1 da nuvem de pontos gerada pelo ARKit.



Fonte: elaborado pelo autor.

Figura 15 – Vista 2 da nuvem de pontos gerada pelo ARKit.



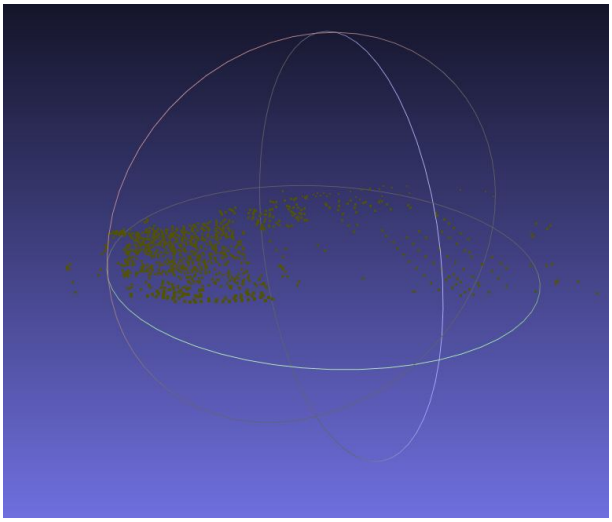
Fonte: elaborado pelo autor.

y da nuvem de pontos estava apresentando um valor menor que os demais e assim os objetos mapeados apresentam esse aspecto. O IPad não foi utilizado nesse caso, pois se acreditou que o erro apresentado em dois dispositivos diferentes seria algum problema da biblioteca e não do dispositivo.

Com os testes iniciais, foram observadas algumas questões sobre o uso da biblioteca ARCore e durante a implementação dessas soluções foi encontrado o trabalho descrito em Martinez (2018), que mesmo utilizado em um contexto diferente se mostrou útil para essa pesquisa. Como visto nesse trabalho, foi desenvolvida uma aplicação capaz de mapear o ambiente em tempo real e usada para simular a colisão de objetos "arremessados" pelo usuário nos objetos mapeados em suas formas de *voxel*. Além disso, também foi avaliado um novo recurso adicionado à biblioteca do ARCore, a *Depth API*.

Alguns dos resultados preliminares utilizando o trabalho de Martinez (2018) pode ser observado na Figura 17, que apresenta os objetos mapeados e sua nuvem de pontos com 3

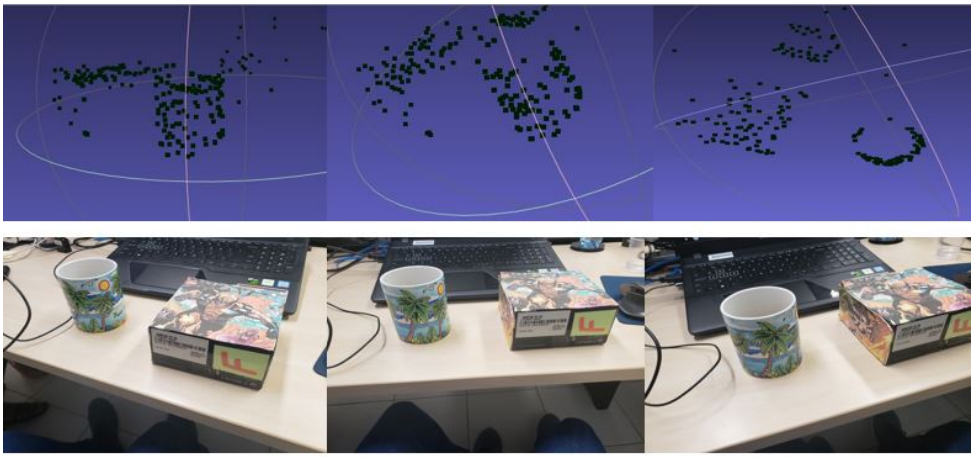
Figura 16 – Nuvem de pontos gerada pelo Wikitude.



Fonte: elaborado pelo autor.

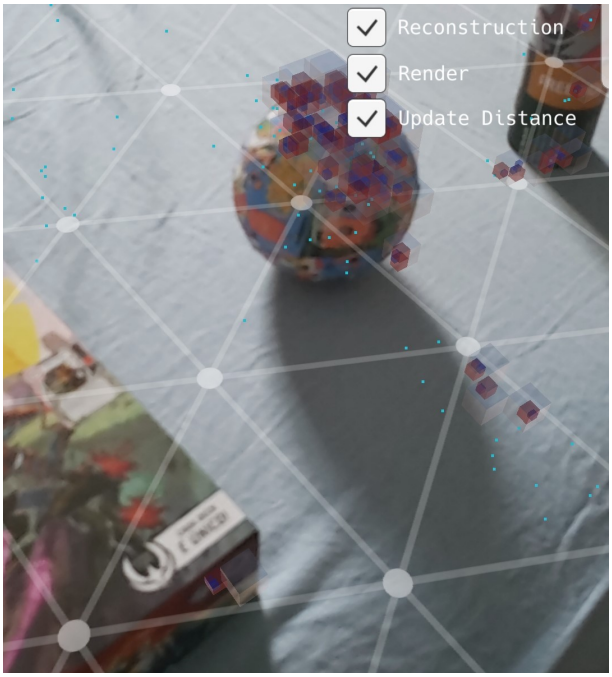
visões diferentes, e nas Figura 18, 19 e 20 com os objetos e pontos mapeados na imagem.

Figura 17 – Experimentos iniciais ARCore, objetos e nuvem de pontos.



Fonte: elaborado pelo autor.

Figura 18 – Experimentos iniciais ARCore, objetos mapeados e pontos detectados exemplo 1.

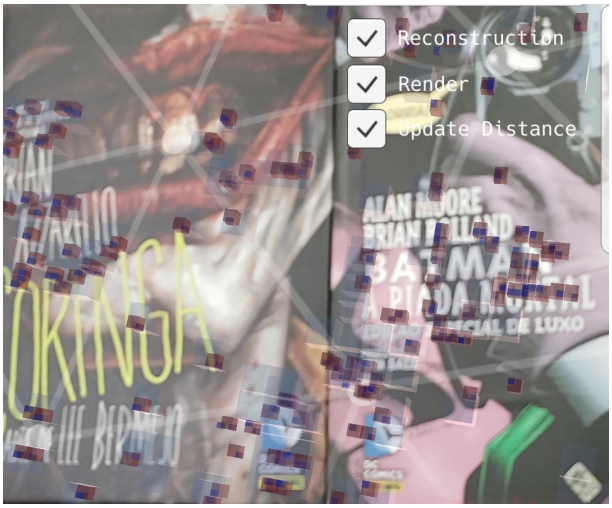


Fonte: elaborado pelo autor.

Figura 19 – Experimentos iniciais ARCore, objetos mapeados e pontos detectados exemplo 2.

Fonte: elaborado pelo autor.

Figura 20 – Experimentos iniciais ARCore, objetos mapeados e pontos detectados exemplo 3.



Fonte: elaborado pelo autor.

3.2 GERAÇÃO DE NUVENS DE PONTOS ESPARSAS

Como apresentado em Martinez (2018), uma das primeiras questões para o uso da nuvem de pontos é encontrar uma forma eficiente para armazenar e acessar os dados. Com o processamento de novos quadros, os pontos antigos e novos são avaliados e é aplicada uma nova verificação para saber se um ponto deve ser adicionado na nuvem. Para resolver tal problema foi utilizada a *Voxel-Hashing* como mostrado em Martinez (2018), visto na equação abaixo:

$$hash(x, y, z) = (x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3) \mod n, \tag{3.1}$$

onde os valores de p_1, p_2 e p_3 são números primos com os valores 73856093, 19349669 e 83492791, respectivamente, e o símbolo $\oplus$ representa o operador OU exclusivo. O valor n representa o tamanho da tabela *hash*. E uma explanação mais detalhada pode ser encontrada em Martinez (2018). Como os valores dos pontos são em ponto flutuante, eles foram convertidos para inteiros utilizando um arrendondamento para ser utilizados na função de *hash*.

3.2.1 Representação por Nível de Detalhe

Após realizar o processo apresentado na seção anterior, aplicando a função de *hash* para os pontos da nuvem, a saída é uma coleção de entrada com um valor de *hash* exclusivo (MARTINEZ, 2018). Mas em alguns casos precisamos de uma representação mais fiel da cena observada, quando o dispositivo fica mais próximo do objeto e em casos quando estamos mais longe do objeto. Com isso, o uso de uma representação adaptativa foi utilizada. A representação de *hash* apresentada anteriormente é utilizada para o primeiro nível de detalhe da reconstrução da cena (o "nível de detalhe 0"). A representação do *hash* é apresentada no trecho de código abaixo.

Listing 3.1 – Código que representa o *hash*.

```
1 struct HashEntry{

3     float position[3];
      int weight;
5     int hash;
      float confidence;
7     float distanceToCamera;
      Voxel voxel;
```

```
9       HashTable LOD[n];
};
```

O código representa a estrutura utilizada para armazenar cada um dos pontos. Ela possui as coordenadas dos pontos 3D (*position*), peso associado ao ponto (*weight*), valor do *hash*, valor de confiança do ponto (*confindence*), distância da câmera do ponto (*distanceToCamera*), o objeto que representa o voxel do ponto 3D (*voxel*) e a estrutura que armazena os demais pontos dos níveis de detalhe (*LOD*).

Dado que foi construído o primeiro nível de detalhe, os outros níveis podem então ser obtidos. O algoritmo utilizado converte o ponto de coordenadas de mundo para coordenadas de *voxel*, chamadas de P_{voxel}. Isso ocorre aplicando a transformação inversa de um ponto, que utiliza a matriz de transformação do *voxel* M_{voxel} e a matriz de transformação do ponto em coordenadas de mundo P_{mundo} (MARTINEZ, 2018). Nesse processo as coordenadas de mundo são normalizadas para porcentagens. Com isso, o *voxel* pai será vai ser divido em grade de *subvoxels* de tamanho iguais, mas esse tamanho é variável podendo ser representados pelos valores: : 2^3, 3^3, 4^3, até n^3 (MARTINEZ, 2018). A função pode ser vista abaixo:

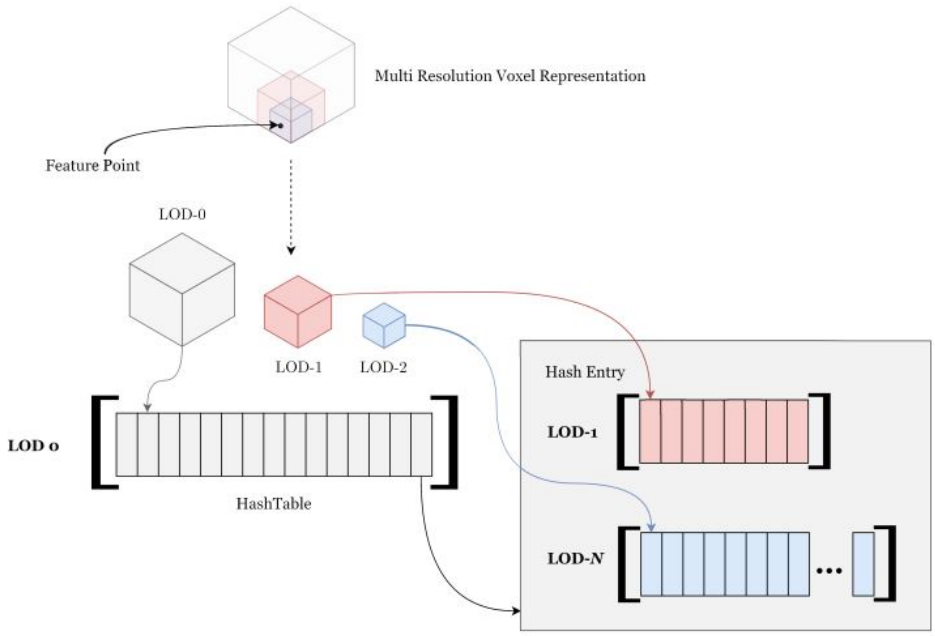
$$P_{voxel} = P_{mundo}^{-1} \cdot M_{voxel} \tag{3.2}$$

Para determinar o *subvoxel* exclusivo de um ponto para o seu nível de detalhe, o *hash* é computado novamente. Dado que a coordenada de um ponto foi normalizada, essas novas coordenadas são dimensionadas utilizando a função 3.3 e depois é aplicada a função 3.1. O procedimento é executado para cada nível de detalhe da reconstrução antes do armazenamento na *hash*. Cada *voxel* tem sua matriz de tabela *hash* que varia de acordo com a resolução no espaço de mundo.

$$P_{escalado} = \left\lfloor \frac{P}{TamanhoGrade} \right\rfloor \cdot TamanhoGrade \tag{3.3}$$

Como citado em Martinez (2018), se um objeto é reconstruído com *voxels* com o nível de detalhe 0 com 10 cm^3, mais dois níveis são definidos, o nível 1 e o nível 2, com 23 e 43 *subvoxels* de tamanhos 5 cm^3 e 2,5 cm^3. Esses novos níveis são salvos na matriz de níveis e seus valores podem ser recuperados utilizando a função *hash*. Um exemplo dessa representação pode ser observado na Figura 21.

Figura 21 – Representação dos níveis de detalhe da representação volumétrica.



Fonte: Martinez (2018).

3.2.2 Valor de Confiança

Como visto no Código 3.1 apresentado, essa estrutura de *hash* apresentada também utiliza o valor de confiança que aumenta conforme a quantidade de vezes que um ponto foi atribuído a um mesmo valor na *hash*. O valor de importância de um voxel é analisado para que o coletor de lixo verifique sua confiança e densidade para remover valores extremos ou que são considerados ruídos (MARTINEZ, 2018).

3.2.3 Operações de *Hash*

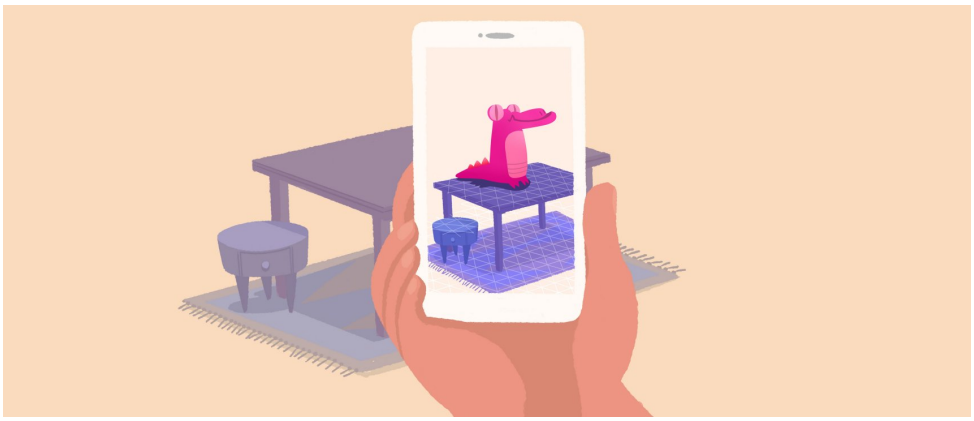
As operações de *hash* utilizadas são: inserção, pesquisa, atualização e exclusão . Durante a inserção são adicionados novos elementos na tabela, mas é realizada uma busca para verificar primeiro o nível 0 e posteriormente os demais níveis são verificados de forma recursiva. Se o *hash* não for encontrado, o *voxel* é inserido na tabela e caso já exista seu valor de confiança é incrementado. Em casos de atualização da tabela, é feita uma busca pelo valor, mas essa busca é linear e tem como base o nível de detalhe da tabela. Essa busca é realizada de acordo com o nível de detalhe. Por fim, no processo de exclusão cada elemento de sua tabela de níveis de detalhe é recuperado e removido para evitar elementos indesejados na tabela ou no dispositivo.

3.2.4 Aplicação

As bibliotecas, tanto ARCore como ARKit, utilizam as âncoras, que são localizações e orientações fixas no mundo real usadas para garantir que os objetos permaneçam no mesmo lugar. Com isso os objetos fixados entregam uma ilusão de que estão ancorados corretamente no mundo real. Nessa aplicação, utilizada nesse trabalho e desenvolvida por Martinez (2018), as âncoras são somente utilizadas nos pontos de nível de detalhe zero, assim todos os *subvoxels* dentro desse espaço são representados como seus filhos. Utilizando esse tipo de abordagem, se uma âncora sofrer algum tipo de mudança de posição ou orientação, todos os elementos contidos naquele local também são transformados.

A primeira etapa do sistema é relacionada à detecção de planos. A aplicação consulta as informações obtidas pelo ARCore e com o deslocamento pelo espaço os novos pontos são detectados e a nuvem de pontos começa a ser gerada. Quando a visualização está disponível, o plano detectado é mostrado na tela do dispositivo como uma malha com uma coloração sendo similar à apresentada na Figura 22. Apenas os pontos observados com confiança maior que um dado limiar podem ser atribuídos ao objeto reconstruído ou cena como visto em Martinez (2018). Alguns dos parâmetros utilizados como informação para reconstrução podem ser alterados na interface do programa, como visto na Figura 23.

Figura 22 – Detecção de planos do ARCore

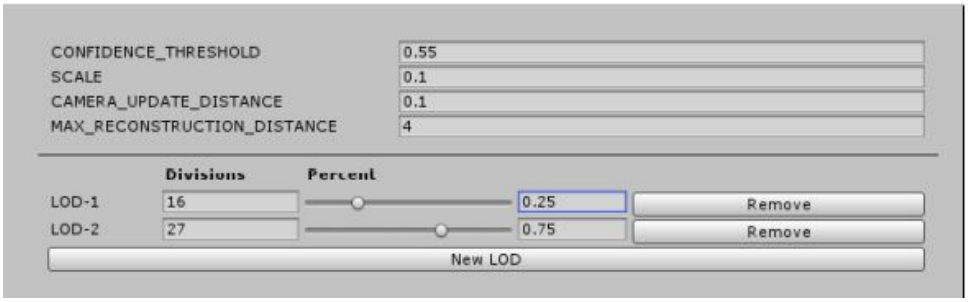


Fonte: https://developers.google.com/ar/discover/concepts.

3.2.5 Interface

Na interface da aplicação, como já apresentado na Figura 23, algumas das variáveis responsáveis por modificar parâmetros podem ser alteradas para assim obter um resultado

Figura 23 – Interface do programa .



Fonte: Martinez (2018).

específico para o contexto que se deseja avaliar. As variáveis, como visto em Martinez (2018, que podem ser modificadas são:

- Limiar de confiança (%): valor que determina quais os *voxels* que serão descartados na etapa de reconstrução. Esse valor pode variar de acordo com a quantidade de vezes que o ponto foi observado;
- Escala (cm^3): representa o tamanho dos *voxels* do nível de detalhe 0. Esse tamanho também afeta o tamanho dos *subvoxels*;
- Distância de atualização (cm): é a distância mínima que o dispositivo deve se mover para causar uma atualização na distância entre cada *voxel* e a câmera;
- Distância máxima de reconstrução (cm): indica qual a distância máxima a ser considerada para que um *voxel* não faça parte da reconstrução;
- Nível de detalhe: representa a informação de quantas resoluções serão construídas.

3.3 GERAÇÃO DE NUVENS DE PONTOS DENSAS

De forma semelhante à geração da nuvem de pontos esparsa, a *Depth API* utiliza apenas informações obtidas a partir da câmera RGB do dispositivo e possui uma lista ainda mais restrita com relação aos modelos de dispositivos compatíveis com o ARCore[2]. Como já comentando anteriormente, essa nova funcionalidade permitiu interações mais realistas com o ambiente e também informações mais precisas com relação aos limites físicos dos objetos.

[2] https://developers.google.com/ar/discover/supported-devices

Segundo as informações fornecidas pela documentação do ARCore[3], o mapa de profundidade é gerado a partir da comparação de múltiplas imagens obtidas e com essa informação é feita a estimativa de distância para cada pixel na imagem. Uma diferença que pode ser observada na documentação é que se o dispositivo possuir algum sensor de profundidade, essa informação é incluída para melhorar a estimativa do mapa final. A informação de profundidade não depende inteiramente de características encontradas a partir de texturas e por isso se obtém melhores resultados se comparado com aplicações que utilizam essas características.

Ainda segundo informações obtidas a partir da página da documentação da *Depth API* para Unity, a representação de cada mapa de profundidade é dada a partir de uma textura 2D. O formato de representação de cada dado na textura é o *Depth16*, possuindo os 3 bits mais significativos como 000. O valor para cada ponto de profundidade é dado com um inteiro de 16 bits sem sinal, com os 13 bits menos significativos representando a distância em milímetros da superfície estimada para o plano de imagem da câmera[4].

3.3.1 Requisitos e limitações da *Depth API*

Para gerar um mapa de profundidade, a API[5] utiliza a informação advinda a partir de duas imagens da cena estáticas. Assim como encontrada na nuvem de pontos esparsa apresentada anteriormente, o mapa de profundidade possui uma confiança associada a cada ponto calculado mas ainda não foi disponibilizada para o público, sua principal diferença é a quantidade de pontos gerados por imagem.

Quando o dispositivo não possuir um sensor de profundidade, os mapas profundidade gerados apenas vão utilizar informação visual, e elementos com baixa textura, como por exemplo paredes brancas, podem apresentar profundidades imprecisas. A profundidade sugerida para estimativa é no intervalo entre 0 e 8 metros, mas a distância sugerida para os objetos visualizados deve ser compreendida entre 0,5 e 5 metros da câmera. O erro pode aumentar conforme a distância avaliada pela câmera aumenta.

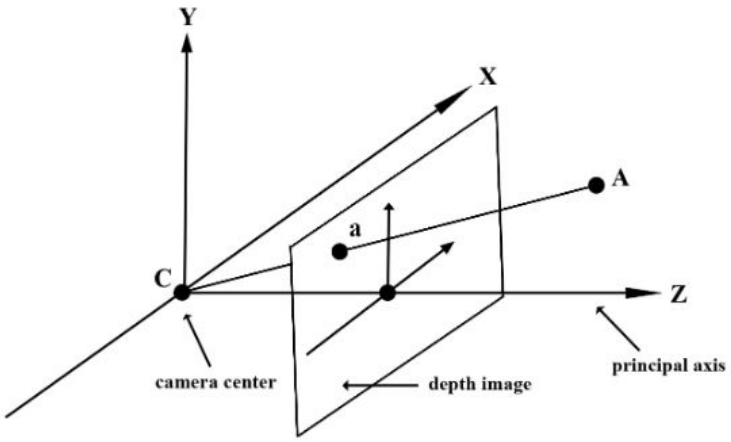
As imagens em RGB processadas pelo dispositivo apresentam uma resolução de 640×480 *pixels*, mas as imagens de profundidade apresentam uma resolução inferior de 160×120 *pixels*. Para cada imagem temos um total de 19.200 pontos que representam cada um dos

[3] https://developers.google.com/ar/develop/unity/depth/overview
[4] https://developers.google.com/ar/develop/unity/depth/overview
[5] https://developers.google.com/ar/develop/unity/depth/overview

pixels obtidos a partir da imagem de profundidade, de forma que ao serem acumulados de forma direta geram uma nuvem de pontos global com uma quantidade de pontos equivalente ao número total de quadros multiplicado por 19.200. O valor da resolução do mapa de profundidade pode variar de acordo com o dispositivo utilizado.

Como visto na Figura 24, um ponto **A** representado em coordenadas de mundo e outro ponto 2D **a** que representa esse mesmo ponto na imagem de profundidade, o valor representado pela API é igual ao comprimento dado por **CA** projetado no eixo principal[6].

Figura 24 – Representação da profundidade da *Depth* API.



Fonte: https://developers.google.com/ar/develop/unity/depth/overview.

3.3.2 Aplicação

Diferente da aplicação utilizada para gerar a nuvem de pontos esparsa acumulada, foi utilizada uma aplicação mais simples que foi obtida a partir do reportório público chamado ARCore *Depth Lab*[7] do trabalho de Du et al. (2020), que oferece um conjunto de exemplos em que o mapa de profundidade pode ser aplicado.

Algumas das aplicações fornecidas reforçam algumas das funcionalidades e possibilidades já mencionadas: locomoção em terreno irregular, verificação de colisão, colisão de partículas de chuva e neve, simulação física, modificação da textura de uma superfície, névoa em RA, oclusões, desfoque de profundidade de campo e mapa de profundidade de cor falsa. O código utilizado é disponibilizado para Android e Unity.

A aplicação utilizada foi a *PointCloud*, que calcula a nuvem de pontos na CPU do dispositivo utilizando a imagem de profundidade. Como já mencionado anteriormente, a

[6] https://developers.google.com/ar/develop/unity/depth/overview
[7] https://github.com/googlesamples/arcore-depth-lab

nuvem de pontos obtida a partir dessa aplicação possui 19.200 pontos para cada imagem utilizada e possui uma resolução menor, que é de 160 por 120 *pixels*, que a obtida na imagem de entrada. Utilizando cada umas dessas nuvens em coordenadas de mundo, criamos uma nova nuvem de forma acumulada, mas esse processo gera muitos pontos redundantes.

3.3.3 Interface

A interface da aplicação é bastante simples e intuitiva, como pode ser visto na Figura 25. Como base foi utilizado o exemplo chamado *DemoCarousel*, que contém uma barra deslizante onde o usuário pode escolher qual aplicação será utilizada. Com a utilização do exemplo, foi possível verificar o funcionamento da *Depth* API com alguns testes simples e observar seu funcionamento através das aplicações presentes nesse exemplo.

Figura 25 – Interface da aplicação utilizando a *Depth* API

Fonte: elaborado pelo autor.

Como apresentado na Figura 25, é possível observar uma barra deslizante onde é permitido navegar entre as aplicações, a aplicação atual selecionada é a *PointCloud* e o botão serve para capturar o mapa de profundidade atual e calcular a nuvem de pontos.

3.4 DETECÇÃO DE PRIMITIVAS

Durante a execução de aplicações utilizando RA, a detecção de algum tipo de primitiva pode tornar a experiência do usuário mais real, pois com o conhecimento da forma do objeto a inserção de conteúdo virtual se torna mais fiel ao ambiente físico. O ARCore utiliza a detecção de planos para melhorar a interação do usuário e mais recentemente

adicionou uma nova API que gera um mapa de profundidade da perspectiva observada na câmera. Com a detecção de primitivas, é possível criar aplicações que podem mapear o ambiente em sua forma mais simples e assim criar um modelo 3D com a informação gerada a partir dessa informação.

Essa seção apresenta uma descrição sobre a técnica de detecção de primitivas utilizada, que aproveita a informação de forma incremental para manter a detecção e rastreamento consistente a partir da nuvem de pontos obtida. Para gerar essa nuvem de pontos foi utilizado o ARCore a partir das aplicações apresentadas anteriormente.

3.4.1 Modelagem Semântica

Como já mencionado anteriormente no Capítulo 2, diferentes tipos de abordagens são utilizadas nesse contexto. Neste trabalho usamos uma abordagem baseada no *Efficient* RANSAC, que foi apresentada em Roberto et al. (2019). Assim como descrito em Schnabel, Wahl e Klein (2007), a extração de primitivas em uma nuvem de pontos tem sua estrutura geral como observado no Algoritmo 1.

Algoritmo 1: Extraindo primitivas de uma nuvem de pontos

```
1  Ψ ← ∅ {Formas extraídas};
2  C ← ∅ {Candidatos};
3  while P(τ, |C|) > p_t do
4      C ← C∪ NovoCandidato();
5      m ← MelhorCandidato(C);
6      if P(|m|, |C|) > p_t then
7          P ← P \ P_m {Remover pontos};
8          Ψ ← Ψ ∪ m;
9          C ← C \ C_m {Remover candidatos inválidos};
10     end
11 end
12 return Ψ;
```

Utilizando como entrada uma nuvem de pontos $\mathcal{P} = \{p_1, ..., p_N\}$ com suas normais associadas $\{n_1, .., n_N\}$, sua saída é um conjunto de primitivas $\Psi = \{\Psi_1, ..., \Psi_n\}$, que tem como conjuntos disjuntos de pontos $\mathcal{P}_{\Psi_1} \subset \mathcal{P}, ..., \mathcal{P}_{\Psi_n} \subset \mathcal{P}$ e possui um conjunto de pontos restantes $\mathcal{R} = \mathcal{P} \setminus \{\mathcal{P}_{\Psi_1}, ..., \mathcal{P}_{\Psi_n}\}$, como apresentado em Schnabel, Wahl e Klein (2007).

Durante cada interação do laço, a primitiva que possuir a pontuação máxima é buscada utilizando a abordagem RANSAC e em seguida novos candidatos são gerados utilizando uma amostragem aleatória de subconjunto mínimos de $\mathcal{P}$ usando a estratégia apresentada em Schnabel, Wahl e Klein (2007). Sempre que novos candidatos são gerados com a maior pontuação m, o melhor candidato só é aceito caso, dado seu tamanho $|m|$ e quantidade de candidatos sorteadas $|\mathcal{C}|$, a probabilidade $\mathcal{P}(|m|, |C|)$ de que nenhum outro candidato foi encontrado durante o intervalo da amostragem for alta o suficiente (SCHNABEL; WAHL; KLEIN, 2007). Esse processo ocorre até que $\mathcal{P}(\tau, |\mathcal{C}|)$, para um tamanho de forma mínimo definido pelo usuário τ, for grande o suficiente.

Um sistema SLAM vai mapear uma área ou objeto observado após obter uma quantidade N de vistas diferentes e com o tempo a quantidade de pontos vai aumentando até que nenhum ponto novo seja adicionado ou até o encerramento da aplicação. Durante um período de tempo, os pontos de uma primitiva podem estar concentrados em regiões cujas texturas são mais visíveis ou regiões que foram observadas primeiro. Em ambos os casos a forma do objeto pode não ser identificada corretamente devido à quantidade insuficiente de informação. Sem o uso da informação temporal, a detecção de primitivas pode não levar em consideração nenhum dos fatores anteriores nem avaliar se dois objetos detectados em sua nuvem de pontos em diferentes instantes estão na verdade representando a informação de um mesmo objeto. As próximas seções são referentes aos novos módulos que foram adicionados ao *Efficient RANSANC* em Roberto et al. (2019) e apresentam soluções para os problemas citados anteriormente.

3.4.2 Fusão de Formas

A fusão de primitivas pode ser utilizada quando os pontos já fornecem toda a informação que precisamos para avaliar se dado duas primitivas as mesmas são objetos diferentes ou partes de um mesmo objeto. Devido à concentração de pontos, um objeto pode ser classificado de forma incorreta como sendo 2 objetos, mas as informações desses objetos podem apresentar um nível alto de semelhança. Essa fusão pode influenciar na avaliação temporal de um objeto e utiliza as seguintes informações para fundir ou não duas primitivas detectadas:

- Planos: dois planos podem ser fundidos se ambos são paralelos dado um limiar de ângulo α_t e se apresentam a uma distância entre eles que é menor que um limiar d_t;

- Esfera: a fusão de esferas usa a informação sobre distância entre seus centros, que deve ser menor que um limiar d_t, e elas também devem apresentar uma diferença entre os raios menor que um limiar r_t.

- Cilindro: nesse caso o critério é o ângulo entre a direção de ambos os eixos, que deve ser menor que um limiar de ângulo α_t, e os cilindros devem possuir uma distância entre si menor que d_t e com uma diferença de raios menor que um limiar r_t.

Segundo Roberto et al. (2019), os valores de d_t e r_t são obtidos em cada caso, já que o tamanho e informações como raios e centros depende do tamanho da primitiva analisada, sendo 2% do maior tamanho das caixas delimitadoras obtidas a partir da nuvem de pontos. Já o α_t é definido como tendo sempre o valor de 5°. A informação referente à distância é outro fator utilizado nesse caso, podendo ampliar ou reduzir os limites da similaridade. Se dois objetos estão distantes, é mais provável que sejam objetos diferentes se comparados com objetos mais próximos. Para os casos onde as primitivas possuem algum tipo de interseção, os limiares são aumentados em 25%, valor obtido de forma experimental.

3.4.2.1 Computação de Parâmetros

Foram apresentados os dados referentes aos critérios de fusão, mas ainda temos que decidir como os parâmetros devem ser computados. Dada uma primitiva obtida a partir da fusão de outras duas no passado, podemos definir que a nova primitiva deve possuir os seus parâmetros como sendo uma média ponderada entres as primitivas que a geraram e seu peso é gerado com base na análise geométrica dos pontos da forma detectada como observado em Roberto et al. (2019). Dado que temos uma nova primitiva gerada pela fusão de outras duas, os novos parâmetros serão definidos através da média ponderada entres as primitivas. A seguinte equação é utilizada para representar os parâmetros da forma fundida P_f :

$$P_f = \sum_{i=1}^{n} w_i P_i, \tag{3.4}$$

onde n representa o número de formas semelhantes a serem fundidas e P_i são seus parâmetros (ROBERTO et al., 2019). Já os pesos w_i são dados pela seguinte equação:

$$w_i = \frac{np}{\sum_{j=1}^{np} dj}. \tag{3.5}$$

Aqui d_j representa a distância Euclidiana entre os pontos e sua projeção na primitiva estimada, np é o número de pontos na nova primitiva, e por fim os pesos são normalizados de forma que $\sum_{i=1}^{n} w_i = 1$ (ROBERTO et al., 2019).

Com a utilização da Equação 3.4, vemos que as formas fundidas vão influenciar nas primitivas resultantes, mas essa média não se aplica para a posição de planos e cilindros. Para a obter a posição correta de um plano, a posição de um ponto é projetada em todos os demais e depois a média ponderada é calculada (ROBERTO et al., 2019). Já no caso de cilindros, será a intersecção dos eixos, só havendo uma modificação no caso de eixos coincidentes ou paralelos, onde sua fusão ocorre aos pares quando existir mais de um (ROBERTO et al., 2019).

3.4.2.2 Critério de Inclusão

Dado que a questão sobre fusão de primitivas já foi explanada, devemos agora observar as primitivas que não foram fundidas. Observando as primitivas restantes, devemos analisar critérios diferentes para avaliar a confiabilidade de uma primitiva, de forma a saber se ela vai ser mantida como primitiva detectada ou não. Com isso em mente, alguns pontos podem ser utilizados para obter essa informação:

- Número de pontos: dado que precisamos de uma grande quantidade de dados para estimar uma primitiva, o número de pontos em uma primitiva confiável é superior a 2% do tamanho total da nuvem (ROBERTO et al., 2019);
- Dispersão: a dispersão dos pontos na primitiva é outra informação importante, já que uma maior densidade de pontos é encontrada em regiões mais texturizadas. Assim a dispersão em formas que são consideradas corretas apresentam um valor menor que 20% do tamanho total da nuvem de pontos (ROBERTO et al., 2019);
- Distância: aqui é usada distância Euclidiana já apresentada. Formas consideradas confiáveis vão apresentar uma distância média menor que 5% da caixa delimitadora da nuvem de pontos completa (ROBERTO et al., 2019);
- Raio: cilindros e esferas visto como confiáveis vão possuir o raio menor que a da caixa delimitadora da nuvem de pontos completa (ROBERTO et al., 2019).

Ao final da análise, são mantidas apenas as primitivas que estão conforme os critérios apresentados anteriormente. Alguns dos dados utilizam como base a dimensão da nuvem de pontos analisada e assim estabelecem os limites dos parâmetros.

3.4.3 Casamento de Formas

Para obter a informação temporal sobre uma primitiva e realizar a verificação da posição da primitiva entre nuvens de pontos diferentes, é utilizada a intersecção entres as caixas delimitadoras e a distância entre os centros de massa. Utilizando a informação do quadro atual, é realizada uma avaliação para identificar a primitiva no quadro passado. Com essa comparação é calculado um escore para cada primitiva que a forma detectada atualmente teve alguma intersecção no quadro anterior. Sua fórmula é dada por:

$$s = \frac{\sum_{i=1}^{k} |\Psi_i| \, p_i \, |\Psi_s|}{\sum_{i=1}^{k} |\Psi_i| \, d_i \sum_{i=1}^{k} |\Psi_i|}, \tag{3.6}$$

onde o valor de ns representa número de índices das primitivas com interseção nos quadros passados, $|\Psi_i|$ é o número de pontos contido na primitiva, p_i e d_i representam a razão da interseção e a distância entre os centros de massa e $|\Psi_s|$ é o número de pontos na primitiva do quadro atual. Uma explicação mais detalhada pode ser encontrada em Roberto et al. (2019)

3.4.4 Atualização e Recuperação de Primitivas

Uma primitiva detectada no quadro atual vai herdar a informação do histórico de primitivas ao qual ela já correspondeu no passado e com esses dados podemos verificar se a atualização está seguindo corretamente (ROBERTO et al., 2019). Se uma primitiva apresenta o mesmo tipo em mais da metade dos quadros, isso incluindo o atual, ela deve ser atualizada utilizando a média ponderada de todas as detecções anteriores, assim a informação temporal vai influenciar durante toda a detecção.

A atualização do novo parâmetro P_u explicada anteriormente é:

$$P_u = \sum_{i=1}^{n-1} w_i P_{u-1} + w_n P_f, \tag{3.7}$$

onde o valor de n é o número de primitivas passadas, isso incluindo a detecção atual, e w_i são seus pesos, sendo eles normalizados. P_{u-1} e P_f são os parâmetros relacionados às detecções anteriores e à atual após a fusão (ROBERTO et al., 2019).

Quando uma primitiva atual tem sua forma diferente do que foi detectado na maior parte do tempo, seu tipo é alterado, mas seus parâmetros são mantidos com o mesmo valor P_u anterior. Assim são avaliadas todas as formas, tanto as detectadas anteriormente quanto as atuais.

3.4.5 Computando a Confiabilidade

Com um conjunto de primitivas detectadas, devemos verificar se elas são confiáveis ou não. Para resolver essa questão, são utilizadas duas informações: a geométrica e a estatística.

Uma primitiva armazena seu histórico, mas mesmo com toda a informação sua forma pode mudar ao longo do tempo devido ao número de vistas observadas ou inconsistências. Para calcular a informação com relação à análise geométrica, utilizamos o peso w_c para cada classe cuja primitiva foi atribuída utilizando a seguinte função:

$$w_c = \frac{1}{\sum_{i=1}^{h} d_i}, \tag{3.8}$$

onde o valor de h representa número de vezes que a classe aparece no histórico e d_i é a distância média dos pontos dessa forma para a nuvem de pontos original (ROBERTO et al., 2019).

Primeiros os pesos são normalizados e os valores são analisados. O que possuir o maior valor será determinado como a classe dominante. As primitivas com um peso superior a 0,75 são consideradas confiáveis, todos as primitivas com pesos menores que 0,5 não são confiáveis, e a confiabilidade das primitivas com pesos entre 0,5 e 0,75 será dada pela análise estatística que verifica o número mínimo de execuções para verificar se uma amostra é aleatória (SHESKIN, 2020). Se a análise mostrar que a classe da primitiva é aleatória, isso avaliado de forma temporal, a primitiva é considerada não confiável.

4 AVALIAÇÃO E RESULTADOS

Neste capítulo serão apresentadas questões relacionadas aos testes realizados, utilizando a nuvem de pontos esparsa e densa, quais os resultados obtidos e uma discussão referente aos pontos observados a partir dos resultados. E ainda questões relacionadas às particularidades das técnicas utilizadas.

4.1 EXPERIMENTOS COM NUVENS DE PONTOS ESPARSAS

Visando avaliar o resultado das aplicações utilizando as bibliotecas selecionadas, no contexto de detecção de primitivas, foi elaborado um experimento com diferentes cenários de teste para identificação das primitivas em cada contexto. Cada caso de teste consiste em realizar a captura da nuvem de pontos durante um intervalo de tempo. Inicialmente o plano da mesa é detectado e depois os dados são armazenados no dispositivo móvel. Para os experimentos foi utilizado um *smartphone* Samsung Galaxy S9. Os dados armazenados são: a pose da câmera fornecida pelo ARCore naquele momento, a imagem disponibilizada pelo ARCore e a nuvem de pontos atual fornecida pela aplicação.

Segundo a informação contida na própria documentação do ARCore, a rotação é representada por um quatérnio e a translação por um vetor contendo três valores e sendo expressa em metros. As imagens disponibilizadas pelo ARCore são no formato YUV 420888, mas são convertidas para escala de cinza. A nuvem de pontos é apresentada como três valores expressos como x, y, z, equivalentes às coordenadas 3D do ponto expressa em metros.

A aplicação roda a 60 quadros por segundo, mas com a adição da etapa para salvar os dados, a aplicação apresentou uma redução nesse tempo de processamento. O conjunto de dados armazenado no dispositivo foi definido com um espaçamento de 60 quadros e cada um desses quadros armazenados é chamado de quadro-chave. Outro dado importante aqui é que o limiar de confiança foi definido para 30%, pois se selecionados valores acima de 50% a nuvem de pontos tem um crescimento menor e mesmo com um espaçamento de 60 quadros os pontos armazenados entre uma nuvem e outra variam pouco.

Mesmo utilizando esse intervalo, a diferença entre a quantidade de pontos entre uma nuvem n e a próxima nuvem era pequena. Em alguns casos o valor chegava apenas a

5 pontos de diferença. Por esse motivo, dentro das nuvens armazenadas, que são salvas a cada 60 imagens, foi feito um novo recorte utilizando o operador de *mod* (módulo da divisão) com um valor 3 para aumentar ainda mais o espaçamento. Assim, por exemplo, considerando nuvens de pontos com índices variando de 0 até 9, seriam escolhidas apenas 4 nuvens (as de índice 0, 3, 6 e 9).

A escala do nível de detalhe zero foi alterada para a menor possível permitida na aplicação, que foi de 1 cm^3, utilizando a movimentação da barra deslizante na tela do aplicativo. Como mencionado no trabalho Martinez (2018), o nível de detalhe zero é relevante para a precisão da representação. Como o foco do trabalho é a detecção de primitivas, precisamos da informação mais detalhada possível. Na representação utilizada, se um *voxel* do nível zero é muito grande, ele pode cobrir um objeto ou região do objeto, fazendo com que sua representação digital seja muito diferente da real. Já distância de atualização foi ajustada para 10 cm e a distância máxima de reconstrução ficou em 4 m.

Visando a detecção de primitivas puramente em um dispositivo móvel, o código de detecção de Roberto et al. (2019) foi adicionado na aplicação como uma DLL, mas seu desempenho foi um pouco inferior ao apresentado em (ROBERTO et al., 2019) que foi uma média de de 343.873 ms por mil pontos processados, pois para o processamento de 500 pontos levou um tempo médio de 348.03 ms e 49,86ms de desvio padrão utilizando a média de 100 execuções. Durante a execução da aplicação com uma quantidade total de pontos entre 500 e 700 pontos, a quantidade de quadros varia entre 20 até 35 por segundo podendo variar de acordo com o aumento do tempo de processamento da detecção.

A Tabela 1 apresenta dados relacionados aos experimentos: a quantidade de primitivas que estão aparecendo em cada um dos casos de teste, a quantidade total de pontos 3D no último quadro de cada um dos casos de testes e a quantidade de quadros-chave avaliada em cada um dos casos de teste. Para fins de visualização, também são exibidos na Figura 26 os cenários montados para cada um dos experimentos, ilustrando o posicionamento das primitivas utilizadas. Também é apresentada a nuvem de pontos final obtida em cada um dos 5 casos de teste na Figura 27.

4.1.1 Resultados

A Figura 28 mostra os resultados obtidos em cada um dos casos de teste. Para esta avaliação, foram utilizadas as mesmas métricas apresentadas em (ROBERTO et al., 2019),

Tabela 1 – Dados dos experimentos.

	Plano(s)	Cilindro(s)	Esfera(s)	Total de pontos	Total de quadros-chave
Caso 1	1	1	1	539	8
Caso 2	2	1	0	682	12
Caso 3	0	2	1	371	13
Caso 4	2	1	1	718	8
Caso 5	3	0	0	564	29

Figura 26 – Caso de teste 1, 2, 3, 4 e 5 (da esquerda para a direita, de cima para baixo). O total de pontos representa a quantidade de pontos na última nuvem.

Fonte: elaborado pelo autor.

que são: precisão (Equação 4.1), cobertura (Equação 4.2) e $F_{0,5}$-Score (Equação 4.3), que é a média harmônica entre precisão e cobertura e com o valor de β sendo 0,5.

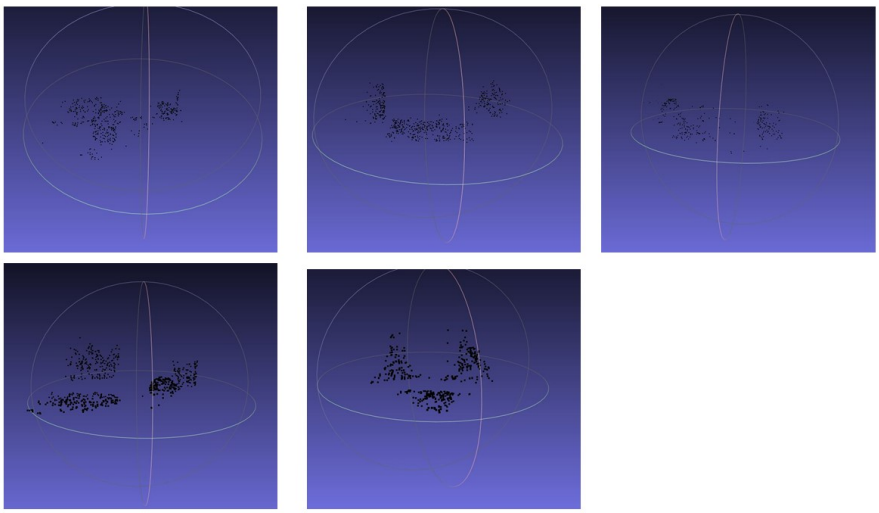
$$P = \frac{VP}{VP + FP} \tag{4.1}$$

$$R = \frac{VP}{VP + FN} \tag{4.2}$$

$$F_{0,5}\text{-Score} = (1 + 0,5^2) \cdot \frac{P \cdot R}{(0,5^2 \cdot P) + R} \tag{4.3}$$

O critério utilizado na classificação para cada um dos casos é o apresentado a seguir:

Figura 27 – Nuvem de pontos final dos teste 1, 2, 3, 4 e 5 (da esquerda para a direita, de cima para baixo).



Fonte: elaborado pelo autor.

1. Falso negativo (FN): nesse caso a avaliação levou em conta uma primitiva que está visível e pode ser observada na nuvem de pontos, mas não foi detectada;

2. Falso positivo (FP): nesse caso a avaliação levou em consideração que foi detectada uma primitiva em uma localização onde ela não existia de fato;

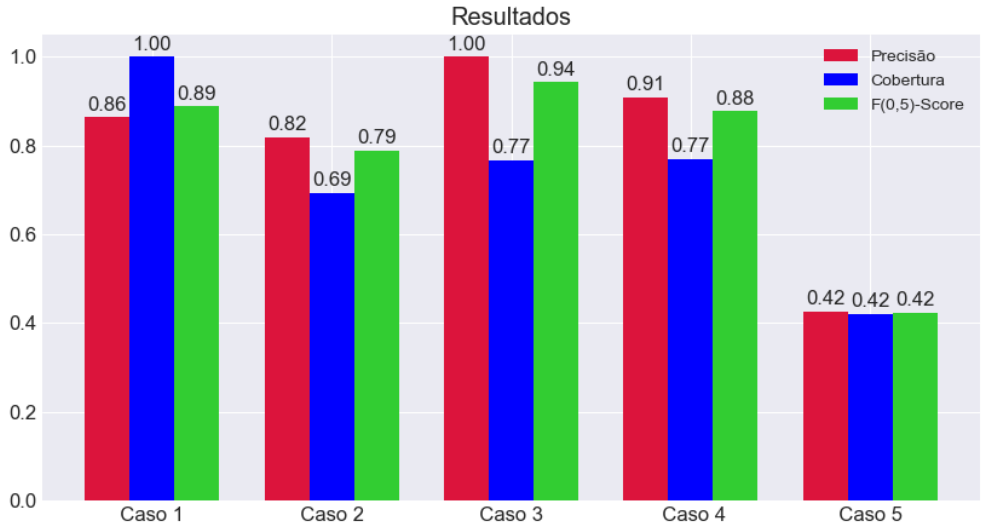
3. Verdadeiro positivo (VP): aqui a avaliação levou em consideração uma primitiva que estava visível na nuvem de pontos e foi classificada corretamente.

Como saída no trabalho de Roberto et al. (2019) são apresentadas para cada uma das primitivas detectadas os pontos pertencentes para aquela primitiva, um ID que presenta o tipo de primitiva ao qual os pontos foram atribuídos e um índice que representa o número da primitiva detectada. Assim os dados apresentados são referentes ao total de primitivas detectadas em cada um dos casos de teste. A métrica $F_{0,5}$-Score foi escolhida por ter um peso mais significativo na detecção de objetos.

4.1.2 Discussão

Nos resultados apresentados, obtivemos uma média aproximada de 80,31% na precisão, mostrando que a técnica utilizada obteve resultados satisfatórios e que esta abordagem conseguiu classificar corretamente um bom número de primitivas, demonstrando a confirmação da pergunta de pesquisa Q1 apresentada. Já na cobertura a média obtida foi de aproximadamente 72,95%, que mostra que essa abordagem conseguiu também avaliar

Figura 28 – Resultados de precisão, cobertura e $F_{0,5}$-Score.



de forma positiva e demonstrando um balanço com relação nos casos de classificação de faltos negativos e positivos, levando em consideração os valores de precisão e cobertura . Mas a técnica incremental acabou favorecendo a ocorrência de falsos negativos.

A técnica utilizada para o mapeamento foi desenvolvida utilizando o Unity e ARCore, e de acordo com a carga utilizada, com relação à atualização da nuvem de pontos e informações do dispositivo, a aplicação pode ter uma redução gradual de quadros por segundo que são processados. Isso evidência umas das limitações relacionadas à pergunta de pesquisa Q2 e logo em seguida será apresentada uma solução para esse problema relacionada à pergunta de pesquisa Q3. Esse problema pode ser solucionado com a migração da plataforma utilizada para o Android nativo e também com a otimização das funções aplicadas para cada um dos cálculos utilizando C++ através da *Native Development Kit (NDK)*.

No caso de teste 5, uma questão com relação à detecção de primitivas foi observada. Esse problema refletiu na precisão, que foi de 42%, ficando abaixo da média dos demais casos de teste. Algumas das primitivas eram detectadas de forma incorreta e em alguns casos a primitiva era classificada corretamente, mas utilizando uma quantidade de pontos extremamente inferior se comparado às demais. Uma das primitivas detectada nesse caso de teste apresentou uma quantidade de pontos inferior a 10 o que não deveria ser considerado uma informação suficiente. Algumas modificações podem ser avaliadas para resolver questões relacionadas ao problema. Uma primeira questão para ser avaliada é qual se-

ria o menor tamanho que uma primitiva poderia possuir para ser considerada confiável, dado que o tamanho da nuvem de pontos gerada no ARCore pode possuir um tamanho significativamente menor.

4.2 EXPERIMENTOS COM NUVENS DE PONTOS DENSAS

Foram realizados dois experimentos contendo as mesmas primitivas utilizadas nos primeiros casos de testes apresentados neste mesmo capítulo. A única modificação feita com relação ao aplicativo base, visto na Figura 25, é que, ao pressionar o botão ‘“Update”, uma nova nuvem de pontos é armazenada no dispositivo no momento que foi pressionada, e também a cada processamento de mais novos 60 quadros, ideia similar a já utilizada nos primeiros experimentos.

Para esses experimentos, foram utilizadas apenas 8 nuvens de pontos, e efetuamos a soma, onde a segunda nuvem é a união da primeira nuvem de pontos com a segunda, a terceira é a soma das nuvens de pontos geradas pela primeira fusão com a terceira nuvem, e assim por diante. Dessa forma a nuvem i vai possuir um total de $i \times 19.200$ pontos, fazendo com que tenhamos na última nuvem um total de 153.600 pontos. Essa abordagem gerar pontos duplicados, mas para esses experimentos nenhuma outra técnica para processar a nuvem de pontos gerada foi avaliada. A quantidade de pontos pode gerar um maior tempo de processamento devido ao tamanho da nuvem de pontos.

A Figura 29 e 30 mostram as primitivas presentes na elaboração dos testes. As métricas utilizadas nesse experimento foram as mesmas apresentadas para a avaliação com a nuvem de pontos esparsa do ARCore.

Figura 29 – Caso de teste 1 *Depth API*.

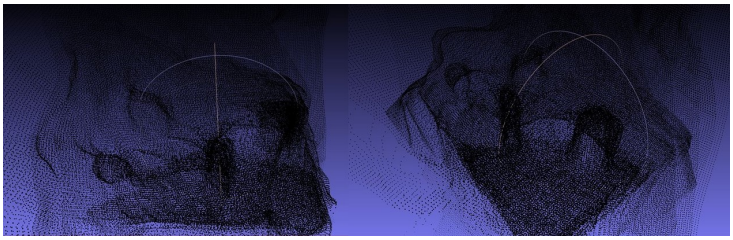
Fonte: elaborado pelo autor.

Figura 30 – Caso de teste 2 *Depth API*.

Fonte: elaborado pelo autor.

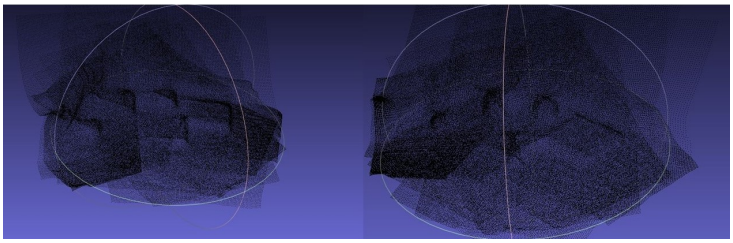
Já as Figuras 31 e 32 apresentam nuvem de pontos acumulada final dos casos de teste 1 e 2 com a *Depth API*, respectivamente, visualizadas de dois pontos de vista diferentes no MeshLab[1].

Figura 31 – Última nuvem de pontos do caso de teste 1 com a *Depth API*.



Fonte: elaborado pelo autor.

Figura 32 – Última nuvem de pontos do caso de teste 2 com a *Depth API*.



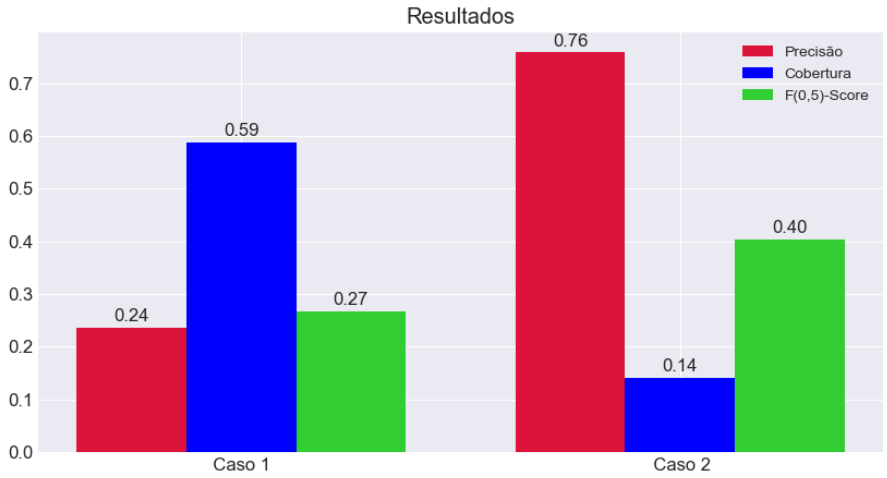
Fonte: elaborado pelo autor.

[1] https://www.meshlab.net/

4.2.1 Resultados e Discussão

A Figura 33 mostra os resultados obtidos nos dois casos de teste utilizando a *Depth API* do ARCore. Os valores médios obtidos para precisão e cobertura foram bastante inferiores se comparados com os primeiros experimentos. A principal diferença observada durante a etapa de avaliação foi o grande número de primitivas classificadas de forma errada, resultando em um grande número de falsos positivos, que foram causados devido ao mapeamento incorreto do ambiente observado. Isso está relacionado às perguntas de pesquisa Q1 e Q2, com alguns problemas associados.

Figura 33 – Resultados da precisão, cobertura e $F_{0,5}$-Score utilizando a *Depth API*

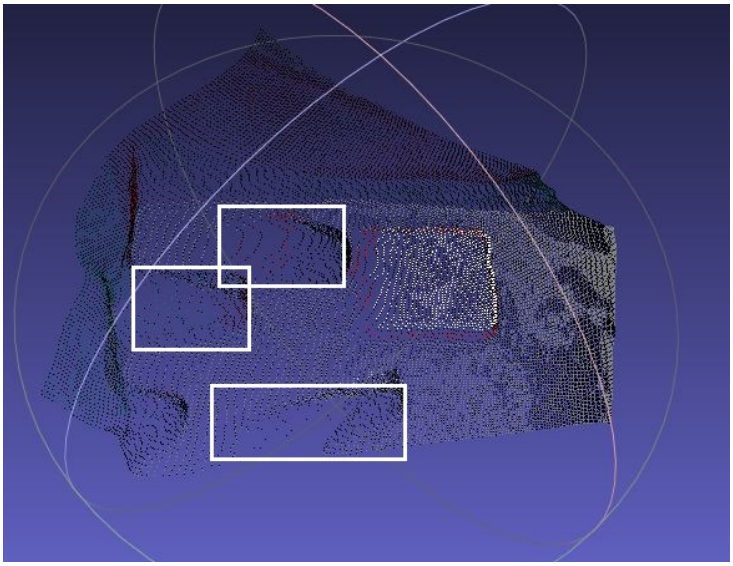


Regiões não visíveis na imagem ou regiões com baixo percentual de confiabilidade acabam gerando mapas de profundidade com deformações ou buracos. Informações imprecisas, quando avaliadas por uma técnica que detecção de primitivas, geram primitivas que não deveriam existir na cena. O problema mais comum observado durante os experimentos foi que planos ficavam curvados e eram detectados como primitivas diferentes.

Outro problema foi o acúmulo de pontos duplicados de forma errônea. Inicialmente a primitiva era identificada corretamente e com o passar do tempo o acúmulo de pontos ao redor do objeto gerou formas distintas e assim a detecção classificou esse objeto de forma diferente. Em outros casos o acúmulo gerou um comportamento já esperado: primitivas que nos mapas iniciais apresentavam uma classificação incorreta tiveram suas formas delimitadas e assim a classificação aconteceu de forma correta. Os casos de erros comentados podem ser observados nas Figuras 34, 35 e 36.

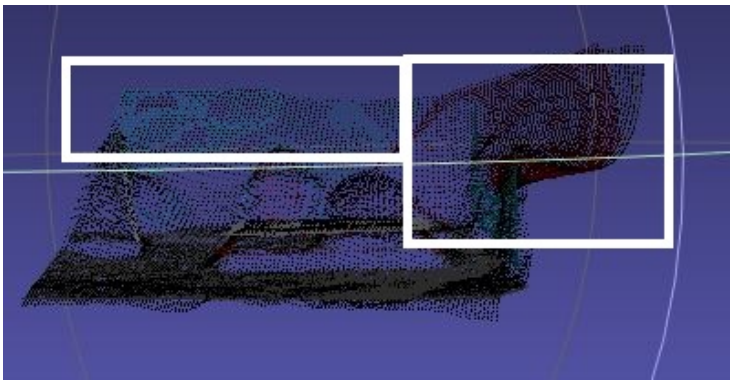
Utilizando o Open3D (2018) foi realizado dois experimentos visando uma redução de

Figura 34 – Regiões deformadas nas nuvens de pontos.



Fonte: elaborado pelo autor.

Figura 35 – Plano deformado.

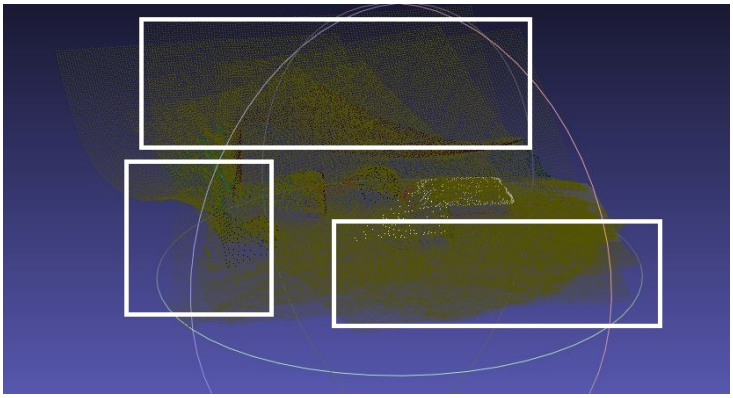


Fonte: elaborado pelo autor.

alguns pontos da nuvem de pontos, principalmente em regiões onde os pontos estavam bastante espaçados que representavam regiões de deformações na nuvem. No primeiro experimento foi avaliada uma abordagem estatística e no segundo uma outra radial. Em ambos os casos era apresentada duas entradas, a quantidade de vizinhos e desvio padrão das distâncias médias no primeiro caso e o número de pontos e o raio quando se utilizada no segundo caso. Utilizando em ambos a variação do segundo parâmetro, foi observado um comportamento bastante similar, com os valores mais altos, de desvio padrão das distâncias médias e raio respectivamente, poucos pontos eram removidos e até um valor mais baixo onde uma grande quantidade muito grande de pontos era removida, mas nesses casos os pontos nas regiões dos objetos eram descartados.

Por apresentar resultados similares foi escolhido a abordagem estatística que apresen-

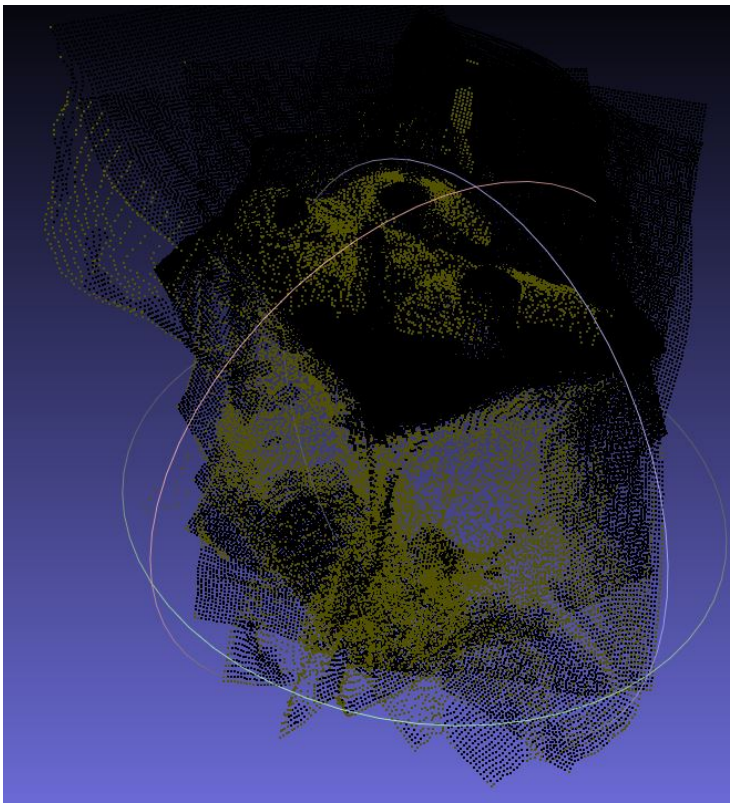
Figura 36 – Acúmulo de erros.



Fonte: elaborado pelo autor.

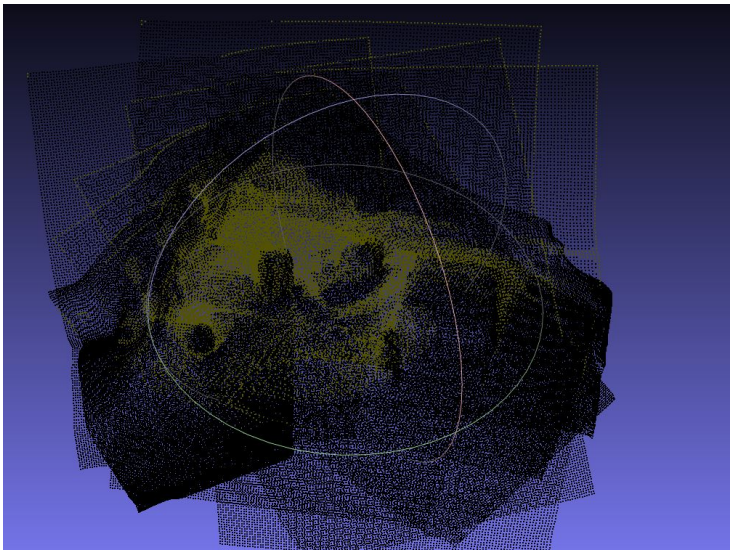
tou como valores finais de entrada 20 vizinhos e um desvio padrão de 0,8. Assim a média de pontos resultante foi de 16.466 de 19.200, esse valor representa 85,39% da nuvem original, mas foi observado principalmente na nuvem final gerada uma grande quantidade de buracos na nuvem e uma remoção significativa de pontos na região dos objetos como mostrado nas Figuras 37 e 30. Nas Figuras os pontos em pretos são os que foram mantidos da nuvem de pontos original e em amarelo os removidos.

Figura 37 – Nuvem de pontos reduzida do Caso de teste 1 *Depth API*.



Fonte: elaborado pelo autor.

Figura 38 – Nuvem de pontos reduzida do Caso de teste 2 *Depth API*.



Fonte: elaborado pelo autor.

5 CONCLUSÕES E TRABALHOS FUTUROS

A pesquisa realizada nesse trabalho permitiu um conhecimento mais aprofundado sobre o uso e limitações das bibliotecas de RA, possibilitando o desenvolvimento de uma aplicação para detecção de primitivas a partir de um dispositivo móvel. Nos experimentos foi demonstrado o uso da biblioteca ARCore e duas de suas formas para o mapeamento, a primeira utilizando a nuvem de pontos esparsa extraídas a partir de objetos texturizados e a segunda com a utilização de mapas de profundidade gerados a partir de imagens RGB capturadas pelo dispositivo. Selecionamos a modificação do algoritmo *Efficient RANSAC* para realizar a detecção de primitivas e a partir dos resultados observamos limitações com relação ao uso do ARCore. Também foram observados aspectos para melhorias nas camadas de mapeamento. Esses aspectos visam refinar a qualidade da nuvem de pontos para melhorar o resultado obtido na detecção e também aspectos relacionados ao tempo de processamento.

A utilização desse tipo de tecnologia permite um desenvolvimento mais rápido de algumas aplicações, mas deve se levar em consideração algumas das limitações das bibliotecas. O ARCore não fornece todas as informações com relação ao coeficiente de distorção da câmera do dispositivo e outras informações não são disponibilizadas ou obtidas de forma trivial, como as transformações aplicadas na imagem utilizadas no dispositivo e acesso às imagens RGB capturadas e processadas, onde as informações utilizadas apenas são invocadas a partir de algum método especificado na documentação da biblioteca.

O entendimento mais aprofundado sobre a biblioteca ARCore demonstrou que essa biblioteca, mesmo sendo mais voltada para o desenvolvimento de aplicativos, pode ser utilizada como parte de soluções voltadas para navegação e entendimento da cena observada em um contexto acadêmico. O fato de utilizar sensores inerciais faz necessário o *hardware* do dispositivo e com a informação obtida a partir da *Depth API* podemos obter mais detalhes do ambiente observado. Outro fator observado é que o ARCore funciona em tempo real (DU et al., 2020) e por esse motivo avaliações ou filtros que seriam utilizados para melhorar a qualidade do mapa de profundidade ou nuvem de pontos devem ser realizados em tempo real ou através de pós-processamento.

Assim como grande parte das técnicas de VSLAM, a detecção de primitivas a partir de nuvens de pontos esparsas apresentada nesse trabalho depende de características detecta-

das nos objetos e se o objeto não possuir textura suficiente o mesmo não será detectado. No ano de 2020, o ARCore apresentou uma API capaz de gerar um mapa de profundidade a partir das imagens RGB de entrada, mas essa API ainda demonstrou algumas limitações para seu uso e também o mapa de profundidade gerado tem uma resolução menor se comparado às imagens RGB capturadas pelo dispositivo, além de poder gerar muitas deformidades em algumas regiões da cena.

Os resultados obtidos durante a etapa de avaliação da técnica mostram que a detecção obteve resultados satisfatórios mas não para os casos utilizando a *Depth API*. Na maioria dos casos as primitivas foram detectadas corretamente, demonstrando assim que, mesmo com uma quantidade de pontos menor que a utilizada em seu trabalho original, o *Efficient* RANSAC modificado apresentado em Roberto et al.(2019) demonstrou eficácia.

Para o caso de teste envolvendo os mapas de profundidade gerados pelo *Depth API*, nem todas as primitivas, quando visíveis, eram detectadas de forma correta, mas tal fato foi devido às deformações ou acúmulos de erro na nuvem de pontos. Os resultados se tornaram inferiores se comparados com os do primeiro experimento realizado.

5.1 CONTRIBUIÇÕES

Esse trabalho apresenta algumas contribuições relevantes em algumas áreas cujo esse trabalho utilizou como apoio. Segue a lista de contribuições:

- Avaliação de três bibliotecas de RA (ARCore, ARKit, Wikitude) com relação ao uso para o mapeamento de primitivas geométricas. Inicialmente foi avaliada a nuvem de pontos gerada por cada uma das bibliotecas no contexto de mapeamento de primitivas geométricas e depois avaliar a detecção de primitivas a partir dessas nuvens;

- Avaliação referente à geração de nuvens de pontos 3D esparsas e densas utilizando a biblioteca ARCore;

- Análise da detecção de primitivas utilizando a biblioteca ARCore a partir de nuvens de pontos esparsas e densas.

5.2 TRABALHOS FUTUROS

Por apresentar novas funcionalidades e com isso novas possibilidades, o uso do ARCore ainda pode ser explorado e analisado em outros contextos. Por exemplo, na reconstrução 3D de objetos com baixa textura, já que informações de profundidade podem ser geradas em tempo real com a nova API do ARCore (DU et al., 2020). Com a delimitação de área adequada a partir dos mapas de profundidade, poderia se realizar uma reconstrução densa de um ou mais objetos.

A migração da aplicação para utilizar o ARCore puramente no Android e não mais no Unity pode trazer grandes ganhos com relação ao desempenho da aplicação. Tal mudança pode acelerar vários dos cálculos efetuados com a utilização do NDK, que permite o uso de código C/C++ no Android. Outro fato é que o SDK do ARCore no Android apresenta informações de forma mais acessível que no Unity.

Uma mudança que também pode melhorar a aplicação é a utilização de um modelo de cliente e servidor, onde a aplicação no dispositivo móvel apenas é responsável por gerar a nuvem de pontos e apresentar os resultados obtidos a partir do processamento no servidor. Essa abordagem pode reduzir o processamento no dispositivo e aumentar a quantidade de dados processada. Com a utilização desse modelo, pode ser avaliada qual a carga de processamento seria ideal para o dispositivo, mas a latência seria outro aspecto para ser avaliado nesse modelo. Esse tipo de arquitetura já se mostrou poderosa em aplicações de reconstrução 3D, como visto em Locher et al. (2016), mas vai necessitar de uma análise de carga para avaliar qual seria o mínimo recomendado de carga computacional para cada um dos lados da aplicação. Uma abordagem interessante seria a avaliação dos serviços oferecidos pela Amazon AWS (SERVICES, 2020) ou Google Cloud (GOOGLE, 2020).

Como a *Depth* API necessita apenas de um dispositivo móvel o uso de *Headsets* onde esse dispositivo é acoplado se tonar possível. Com as informações disponibilizadas por essa nova API podemos utilizar o *smartphone* em contextos onde antes só era possível a utilização de *Headsets* como: Oculos Rift, HTC Vive e Samsung HMD Odyssey.

Outro trabalho importante seria uma nova análise de novas técnicas focadas para detecção de primitivas. Muitas abordagens atuais estão utilizando aprendizagem profunda e apresentando resultados promissores em diferentes áreas. Com o uso de uma aplicação que utiliza um modelo de cliente e servidor, o poder de processamento fica mais distribuído, permitindo a utilização de mais técnicas.

Com essa análise foi possível observar que a utilização da *Depth API* sem nenhum tipo de tratamento não gera um resultado confiável. Abaixo são apresentadas algumas abordagens que podem ser exploradas para melhoria na qualidade da nuvem resultante:

1. Filtro de confiabilidade: realizar uma avaliação prévia para selecionar apenas pontos com nível de confiabilidade acima de um limiar confiável. Tais valores deveriam ser escolhidos de acordo com a aplicação onde essa informação será utilizada;

2. Processamento externo: devido ao grande número de dados que estão sendo gerados no dispositivo, uma boa opção seria a utilização de um servidor capaz de processar esses dados e retornar apenas as informações relevantes, tais como: posição, orientação, raio e normais dos objetos.

As duas abordagens avaliadas mostram o potencial de uso do ARCore, mas para serem utilizadas em aplicações mais robustas e com requisitos mais exigentes, algum processamento adicional será necessário. Com uma integração simples, a *Depth API* poderia fazer parte de um sistema para a navegação de um robô, utilizando as informações advindas do mapa de profundidade, ou na geração de mapas de profundidade de uma região para o mapeamento da área. Os projetos apresentados no repositório do *Depth Lab* são um bom ponto de partida para análise de possíveis aplicações a partir dessa nova API.

REFERÊNCIAS

AZUMA, R. T. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, MIT Press, v. 6, n. 4, p. 355–385, 1997.

CADENA, C.; CARLONE, L.; CARRILLO, H.; LATIF, Y.; SCARAMUZZA, D.; NEIRA, J.; REID, I.; LEONARD, J. J. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, IEEE, v. 32, n. 6, p. 1309–1332, 2016.

CHEN, Y.; DAVIS, T. A.; HAGER, W. W.; RAJAMANICKAM, S. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, ACM New York, NY, USA, v. 35, n. 3, p. 1–14, 2008.

DAVIS, T. A.; GILBERT, J. R.; LARIMORE, S. I.; NG, E. G. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, ACM New York, NY, USA, v. 30, n. 3, p. 353–376, 2004.

DAVISON, A. J.; REID, I. D.; MOLTON, N. D.; STASSE, O. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 29, n. 6, p. 1052–1067, 2007.

DILEK, U.; EROL, M. Detecting position using arkit. *Physics Education*, IOP Publishing, v. 53, n. 2, p. 025011, 2018.

DU, R.; TURNER, E.; DZITSIUK, M.; PRASSO, L.; DUARTE, I.; DOURGARIAN, J.; AFONSO, J.; PASCOAL, J.; GLADSTONE, J.; CRUCES, N.; IZADI, S.; KOWDLE, A.; TSOTSOS, K.; KIM, D. DepthLab: Real-time 3D Interaction with Depth Maps for Mobile Augmented Reality. In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. [S.l.]: ACM, 2020. (UIST).

ENGEL, J.; SCHÖPS, T.; CREMERS, D. Lsd-slam: Large-scale direct monocular slam. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 834–849.

FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, ACM, v. 24, n. 6, p. 381–395, 1981.

GARTNER, I. a. i. a. *Gartner Hype Cycle*. 2018. Disponível em: <https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/>.

GOOGLE. *Google Cloud*. 2020. Disponível em: <https://cloud.google.com/?&utm_source=google&utm_medium=cpc&utm_campaign=latam-BR-all-pt-dr-bkws-all-all-trial-e-dr-1009133-LUAC0010101&utm_content=text-ad-none-none-DEV_c-CRE_452834960568-ADGP_BKWS+%7C+Multi+~+Google+Cloud-KWID_43700047045899968-kwd-301173107424-userloc_1001624&utm_term=KW_google%20cloud-ST_Google+Cloud&gclid=EAIaIQobChMIxMKLpIKK7AIVAwiRCh2Sywj9EAAYASAAEgLAefD_BwE&gclsrc=aw.ds>.

ILA, V.; POLOK, L.; SOLONY, M.; SVOBODA, P. Slam++-a highly efficient and temporally scalable incremental slam framework. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 36, n. 2, p. 210–230, 2017.

JINYU, L.; BANGBANG, Y.; DANPENG, C.; NAN, W.; GUOFENG, Z.; HUJUN, B. Survey and evaluation of monocular visual-inertial slam algorithms for augmented reality. *Virtual Reality & Intelligent Hardware*, Elsevier, v. 1, n. 4, p. 386–410, 2019.

KAESS, M.; JOHANNSSON, H.; ROBERTS, R.; ILA, V.; LEONARD, J. J.; DELLAERT, F. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, Sage Publications Sage UK: London, England, v. 31, n. 2, p. 216–235, 2012.

KAESS, M.; RANGANATHAN, A.; DELLAERT, F. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, IEEE, v. 24, n. 6, p. 1365–1378, 2008.

KAISER, A.; ZEPEDA, J. A. Y.; BOUBEKEUR, T. A survey of simple geometric primitives detection methods for captured 3d data. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2019. v. 38, n. 1, p. 167–196.

KLEIN, G.; MURRAY, D. Parallel tracking and mapping for small ar workspaces. In: IEEE. *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. [S.l.], 2007. p. 225–234.

LEUTENEGGER, S.; LYNEN, S.; BOSSE, M.; SIEGWART, R.; FURGALE, P. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, SAGE Publications Sage UK: London, England, v. 34, n. 3, p. 314–334, 2015.

LI, D.; FENG, C. Primitive fitting using deep boundary aware geometric segmentation. *arXiv preprint arXiv:1810.01604*, 2018.

LI, M.; MOURIKIS, A. I. Improving the accuracy of ekf-based visual-inertial odometry. In: IEEE. *2012 IEEE International Conference on Robotics and Automation*. [S.l.], 2012. p. 828–835.

LOCHER, A.; PERDOCH, M.; RIEMENSCHNEIDER, H.; GOOL, L. V. Mobile phone and cloud—a dream team for 3d reconstruction. In: IEEE. *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. [S.l.], 2016. p. 1–8.

LUKÁCS, G.; MARTIN, R.; MARSHALL, D. Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. In: SPRINGER. *European conference on computer vision*. [S.l.], 1998. p. 671–686.

MARTINEZ, M. G. *Smartphone 3D Reconstruction for Physical Interactions in Augmented Reality*. Dissertação (Mestrado) — Trinity College, sep 2018.

MATTAUSCH, O.; PANOZZO, D.; MURA, C.; SORKINE-HORNUNG, O.; PAJAROLA, R. Object detection and classification from large-scale cluttered indoor scans. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2014. v. 33, n. 2, p. 11–21.

MOHAMED, S. A.; HAGHBAYAN, M.-H.; WESTERLUND, T.; HEIKKONEN, J.; TENHUNEN, H.; PLOSILA, J. A survey on odometry for autonomous navigation systems. *IEEE Access*, IEEE, v. 7, p. 97466–97486, 2019.

MOURIKIS, A. I.; ROUMELIOTIS, S. I. A multi-state constraint kalman filter for vision-aided inertial navigation. In: IEEE. *Proceedings 2007 IEEE International Conference on Robotics and Automation*. [S.l.], 2007. p. 3565–3572.

MUR-ARTAL, R.; TARDÓS, J. D. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, IEEE, v. 33, n. 5, p. 1255–1262, 2017.

MUR-ARTAL, R.; TARDÓS, J. D. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, IEEE, v. 2, n. 2, p. 796–803, 2017.

NEWCOMBE, R. A.; LOVEGROVE, S. J.; DAVISON, A. J. Dtam: Dense tracking and mapping in real-time. In: IEEE. *2011 international conference on computer vision*. [S.l.], 2011. p. 2320–2327.

NIANTIC INC., N. *Pokémon GO*. 2019. Disponível em: <https://www.pokemongo.com/pt-pt/>.

OESAU, S.; LAFARGE, F.; ALLIEZ, P. Planar shape detection and regularization in tandem. In: WILEY ONLINE LIBRARY. *Computer Graphics Forum*. [S.l.], 2016. v. 35, n. 1, p. 203–215.

QIN, T.; LI, P.; SHEN, S. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, IEEE, v. 34, n. 4, p. 1004–1020, 2018.

ROBERTO, R.; LIMA, J. P.; UCHIYAMA, H.; TEICHRIEB, V.; TANIGUCHI, R.-i. Geometrical and statistical incremental semantic modeling on mobile devices. *Computers & Graphics*, Elsevier, v. 84, p. 199–211, 2019.

ROBERTO, R. A.; UCHIYAMA, H.; LIMA, J. P. S.; NAGAHARA, H.; TANIGUCHI, R.-i.; TEICHRIEB, V. Incremental structural modeling on sparse visual slam. *IPSJ Transactions on Computer Vision and Applications*, Springer, v. 9, n. 1, p. 5, 2017.

SALAS-MORENO, R. F.; NEWCOMBE, R. A.; STRASDAT, H.; KELLY, P. H.; DAVISON, A. J. Slam++: Simultaneous localisation and mapping at the level of objects. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2013. p. 1352–1359.

SCHNABEL, R.; WAHL, R.; KLEIN, R. Efficient ransac for point-cloud shape detection. In: WILEY ONLINE LIBRARY. *Computer graphics forum*. [S.l.], 2007. v. 26, n. 2, p. 214–226.

SERVICES, A. W. *Amazon Web Services (AWS)*. 2020. Disponível em: <https://aws.amazon.com/pt/>.

SHESKIN, D. J. *Handbook of parametric and nonparametric statistical procedures*. [S.l.]: crc Press, 2020.

SUALEH, M.; KIM, G.-W. Simultaneous localization and mapping in the epoch of semantics: a survey. *International Journal of Control, Automation and Systems*, Springer, v. 17, n. 3, p. 729–742, 2019.

TAKETOMI, T.; UCHIYAMA, H.; IKEDA, S. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, Springer, v. 9, n. 1, p. 16, 2017.

THIERY, J.-M.; GUY, É.; BOUBEKEUR, T. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics (TOG)*, ACM, v. 32, n. 6, p. 178, 2013.

UENOHARA, M.; KANADE, T. Vision-based object registration for real-time image overlay. *Computers in Biology and Medicine*, Elsevier, v. 25, n. 2, p. 249–260, 1995.

ZHANG, X.; YAO, X.; ZHU, Y.; HU, F. An arcore based user centric assistive navigation system for visually impaired people. *Applied Sciences*, Multidisciplinary Digital Publishing Institute, v. 9, n. 5, p. 989, 2019.

ZHOU, F.; DUH, H. B.-L.; BILLINGHURST, M. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In: IEEE COMPUTER SOCIETY. *Proceedings of the 7th IEEE/ACM international symposium on mixed and augmented reality*. [S.l.], 2008. p. 193–202.

ZHOU, Q.-Y.; PARK, J.; KOLTUN, V. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

ZHOU, Y.; YIN, K.; HUANG, H.; ZHANG, H.; GONG, M.; COHEN-OR, D. Generalized cylinder decomposition. *ACM Trans. Graph.*, v. 34, n. 6, p. 171–1, 2015.