PROJETO ARL: UMA BIBLIOTECA PARA COMPARTILHAMENTO DE OBJETOS UTILIZANDO REALIDADE AUMENTADA

Maicon Santos da Silva, Dalton Solano dos Reis – Orientador

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

maiconsilva@furb.br, dalton@furb.br

Resumo: *Este artigo apresenta o desenvolvimento de uma biblioteca para dispositivos móveis Android, que permite outros desenvolvedores construírem aplicações para o compartilhamento de objetos 3D entre seus usuários utilizando realidade aumentada. A construção da biblioteca é dívida em duas partes: a primeira, onde fica implementado todo o funcionamento do projeto e utiliza a biblioteca ARCore da Google, desenvolvida em Unity e a segunda, fazendo o encapsulamento do projeto em formato de biblioteca, desenvolvida no Android Studio. Para validar o funcionamento da biblioteca foi construído uma aplicação Android utilizando suas funcionalidades. Foram realizados diversos testes de mesa cadastrando objetos em diferentes pontos geográficos na cidade. A aplicação se apresentou sensível à precisão dos sensores do hardware utilizado (GPS, acelerômetro, giroscópio e bússola), podendo apresentar falhas dependendo de quão distante estão os objetos detectados e visualizados pelo usuário final da aplicação.*

Palavras-chave*: Realidade aumentada. Geolocalização. ARCore. Biblioteca Android.*

1 INTRODUÇÃO

A computação gráfica, atualmente, está presente em diversas áreas, desde engenharias e medicina até jogos e animações. Segundo Manssour e Cohen (2006, p. 1), ela "é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a geração (síntese) de imagens através do computador". Dentre suas possibilidades, softwares de simulação têm um papel importante pois possibilitam analisar informações de forma mais clara sem precisar de fato pôr em prática determinado processo, como por exemplo na área da medicina para treinamento de cirurgias, na engenharia civil para construção de edifícios ou até mesmo em simuladores para treinamento de pilotos de aeronaves.

Dentro do mundo da simulação há a Realidade Aumentada (RA), um conceito que existe há bastante tempo, mas que só teve um crescimento significativo nos últimos anos. Segundo Azuma (2001, p. 1, tradução nossa), "Um sistema de RA complementa o mundo real com objetos virtuais (gerados por computador) que parecem coexistir no mesmo espaço que o mundo real" e segundo o autor deve possuir as seguintes propriedades: combinar objetos reais e virtuais em um ambiente real; rodar de forma interativa e em tempo real; e registrar (alinhar) objetos reais e virtuais uns com os outros.

No Brasil, a RA surgiu durante a década de 90 num movimento de novas tecnologias envolvendo áreas multidisciplinares: computação gráfica, sistemas distribuídos, computação de alto desempenho, sistemas de tempo real, interação humano-computador, periféricos, entre outros. Atualmente vários projetos de RA existem em diversas áreas, tais como na cartografia (SOUZA, 2016), publicidade (LUTFI; RAPOSO, 2010), arquitetura, saúde (FARIAS, 2015), jogos (KIRNER, 2011), entre outros.

Atualmente a convergência tecnológica permite o desenvolvimento de aplicações de RA que envolvem muitos recursos trabalhando juntos, tais como: Global Position System (GPS), acelerômetros, câmeras, telefones celulares, notebooks, entre outros (KIRNER, 2011, tradução nossa). Porém, o problema que existe é que a manipulação dessas tecnologias e, principalmente, da RA depende de programadores e conhecimento técnico.

Diante do exposto, este projeto desenvolveu uma biblioteca para a construção de aplicações que permitem posicionar virtualmente, no mundo real, objetos modelados pelo usuário e importados na aplicação; e, ainda, compartilhar estes objetos entre os usuários finais, já que abstrai o repositório a ser utilizado. O projeto não foi desenvolvido direcionado para nenhuma área específica, suas funcionalidades podem ser usadas para diversos propósitos, deixando livre para uso da criatividade do desenvolvedor da aplicação final.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentadas algumas tecnologias, conceitos e algoritmos utilizados no desenvolvimento desta biblioteca. Serão apresentados os seguintes assuntos: conceito de realidade aumentada, sensores de geolocalização e arquivos Wavefront (`obj`). Por fim, são apresentados os trabalhos correlatos a este trabalho.

2.1 REALIDADE AUMENTADA

A realidade aumentada é um sistema que suplementa o mundo real com objetos virtuais (gerados por computador) que parecem coexistir no mesmo espaço no mundo real (Azuma, 2001). Segundo Azuma, a realidade aumentada precisa seguir três propriedades: combinar objetos reais e virtuais num ambiente real; rodar interativamente, e em tempo real; e registrar objetos reais e virtuais uns com os outros.

Para reproduzir RA alguns fatores são necessários. Fazem parte do processo da RA a visualização, rastreamento, registro e calibragem. No campo da visualização, podemos classificar a exibição da realidade aumentada em três categorias: Head Worn (óculos), portátil e projetivo.

a) Head-Worn Displays (HWD): nessa categoria os usuários montam a visualização em suas cabeças, montando a imagem em seus olhos através de óculos. Nesses dispositivos é possível visualizar objetos virtuais em tempo real e visualização de vídeo. A visualização de vídeo consiste em capturar o vídeo do ambiente, aplicar o RA e exibir, diferente de visualizar em tempo real que aplica a RA enquanto o usuário visualiza o ambiente;
b) visualização portátil: alguns sistemas de RA usam dispositivos portáteis, painéis de LCD (celulares, tablets, entre outros) que possuem câmeras alocadas para poder renderizar objetos virtuais sob o mundo real;
c) visualização projetiva: nesse caso, a visualização da RA é diretamente aplicada ao ambiente físico. O objetivo é colocar os objetos projetados literalmente no mesmo plano que o objetos do mundo real. Geralmente ocorre usando algum tipo de projetor, sem a necessidade do uso de óculos ou qualquer dispositivo portátil. Embora também seja possível trabalhar com visualização projetiva usando óculos, só que nesse caso as imagens são projetadas através da vista do usuário em objetos do mundo real, ou seja, os objetos são revestidos com material retrorrefletivo que refletem luz de volta. Neste caso, usuários podem ver diferentes imagens no mesmo alvo.

Alguns problemas são identificados na área de visualização de RA. Visualizar através de óculos acaba não tendo brilho, resolução, campo de visão e contraste suficientes. E por causa desses pontos, objetos virtuais não conseguem sobrepor totalmente objetos do mundo real, fazendo com que a visualização fique misturada. Outro problema também é o fato de a câmera ser montada longe da localização do olho, então o que o olho consegue ver acaba sendo diferente do capturado pela câmera.

A área do rastreamento é um campo crucial da RA, sendo responsável por identificar a orientação e posição da vista do usuário. Nos dias de hoje, os avanços tecnológicos permitem que muitos sistemas tenham resultados muito precisos, utilizando técnicas híbridas de rastreamento com acelerômetro e rastreamento em vídeo. É possível também aprimorar o rastreamento com marcadores físicos no mundo real que são colocados como locais conhecidos no mundo real, esse é o conceito de rastreamento visual.

Segundo Azuma (2001, p. 36), ao trabalhar com rastreamento em ambientes não preparados ou com RA em dispositivos móveis não é muito prático trabalhar com marcadores e por isso outras abordagens devem ser tomadas, como por exemplo o uso de acelerômetro, giroscópio, processamento de imagem ou GPS. Na hora de registrar um objeto virtual no mundo real é necessário ainda passar pelo processo de calibragem que é composto por vários fatores como parâmetros da câmera, campo de visão, sensores de deslocamento, localização dos objetos, distorções, entre outros. Hoje já existem várias técnicas para auto calibragem e em muitos casos é possível evitar essa etapa.

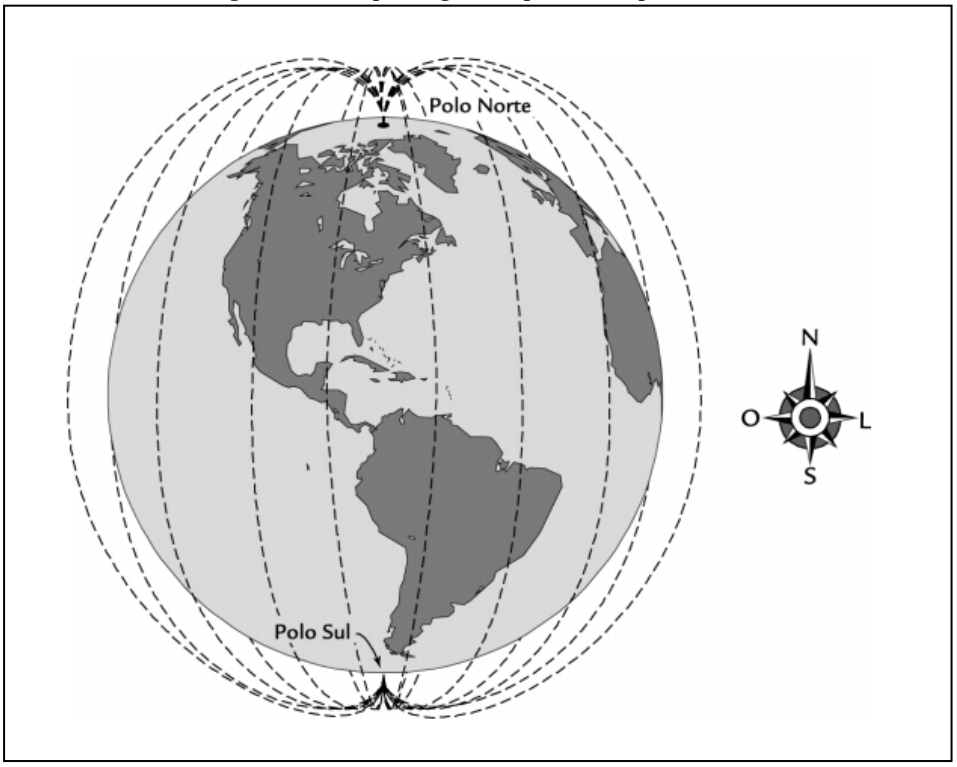
2.2 BÚSSOLA

A bússola foi a invenção tecnológica mais importante desde a roda, mesmo após 700 anos de sua criação, ainda é utilizada em embarcações (ACZEL, 2001). Segundo Aczel (2001, p. 8), ela:

> [...] funciona porque a Terra é um magneto gigante. Um magneto é um objeto que induz um campo magnético; esse campo é uma região do espaço em torno do magneto dentro da qual existem linhas invisíveis de força que correm entre dois pontos denominados os polos norte e sul do magneto. Um campo magnético é induzido cada vez que elétrons se movem, como quando uma corrente elétrica flui. Magnetos naturais, como a magnetita, derivam seu magnetismo da maneira peculiar como os elétrons se movem dentro deles. O campo magnético exerce uma atração sobre o ferro e elementos similares e pode atrair ou repelir outros magnetos, dependendo da orientação destes. Polos iguais se repelem; polos diferentes se atraem. Se introduzirmos um magneto num campo magnético, ele vai, desde que possa se mover livremente, alinhar-se com o novo campo magnético.

A bússola é composta por uma agulha, um magneto que fica suspenso no ar ou na água e é capaz de girar livremente. Essa agulha reage ao campo magnético da Terra (Figura 1) e se alinha a ele, assim indicando a direção do norte magnético. Apesar de ser uma ferramenta confiável, a bússola sofre com alguns fatores, como o desvio do norte magnético em relação ao verdadeiro polo norte geográfico e por ser uma ferramenta de detecção magnética, com a presença de objetos magnéticos por perto.

Figura 1 – Campo magnético produzido pela Terra



Fonte: Aczel (2001).

Segundo Setzer (2014), as bússolas presentes nos dispositivos móveis também sofrem com os problemas presentes nas bússolas tradicionais. Com objetivo de melhorar a precisão ao determinar o norte da Terra, as bússolas contam o auxílio do acelerômetro. O acelerômetro prove informação da posição do dispositivo no espaço, essa informação combinada com os dados da bússola provem uma melhor precisão na detecção do norte magnético.

2.3 FORMATO DE ARQUIVO *OBJECT FILE* (.OBJ)

Segundo McHenry e Bajcsy (2008, p. 12), arquivos `obj` são arquivos de texto desenvolvidos pela Wavefront Technologies (hoje em dia, Alias/Wavefront) e que são adotados por diversos softwares de computação gráfica para a construção de modelos 3D. Os formato consiste em linhas, onde cada uma possui uma chave e vários valores. Essa chave é que indica o tipo de informação que será lida. Na Quadro 1 está descrito uma lista de chaves que podem ser utilizadas dentro de um arquivo `obj`.

Quadro 1 – Lista de algumas das chaves que podem ser usadas dentro de um arquivo `obj`

Chave	Descrição
#	Comentário
v	Vértice
l	Linha
f	Superfície
vt	Coordenada de textura
vn	Normal
g	Grupo
...	...

Fonte: McHenry e Bajcsy (2008).

Arquivos obj não armazenam informação de cores, mas podem fazer referência a materiais que serão importados num arquivo separado. Os arquivos responsáveis por levar a informação de cores são chamados de mtl e podem ser chamados utilizando a chave mitlib. Arquivos mtl armazenam informação RGB para cor, refração, reflexo, entre outras propriedades.

2.4 TRABALHOS CORRELATOS

Foram escolhidos três trabalhos correlatos que possuem características semelhantes aos objetivos do estudo proposto. O primeiro é um aplicativo para dispositivo móvel que permite posicionar modelos 3D e visualizar com realidade aumentada (AUGMENT, 2018), descrito no Quadro 2. No Quadro 3 é apresentado o jogo Pokémon GO para dispositivo móvel, baseado na série animada de televisão Pokémon (PRADO, 2016). No Quadro 4 é descrito uma rede social para compartilhamento de imagens e vídeos que permite usar filtros e posicionar objetos no cenário usando realidade aumentada (ALECRIM, 2016).

Quadro 2 – Augment: visualizador 3D de modelos

Referência	Augment (2018)
Objetivos	Permitir que o usuário importe modelos 3D personalizados e o posicione em cena para visualização com realidade aumentada.
Principais funcionalidades	Oferecer um ambiente onde é possível importar qualquer tipo de modelo e posicioná-los em cena para visualizar utilizando realidade aumentada. Permite também que objetos sejam cadastrados junto com imagens como marcador.
Ferramentas de desenvolvimento	Não foi encontrado em qual linguagem a ferramenta foi desenvolvida. Sabe-se que utilizou as bibliotecas OpenGL e Vuforia.
Resultados e conclusões	Por ser um produto comercial, não foram encontrados resultados e dados oficiais sobre o produto.

Fonte: elaborado pelo autor.

O produto Augment é dividido em três plataformas: o Augment Manager, o aplicativo móvel principal e o Augment Desktop. O Augment Manager é uma plataforma web que permite visualizar todas as configurações da conta, fazer upload de modelos e configurar informações sobre ele (como nome, unidade de medida, se o objeto será público ou privado, entre outras opções). Também é possível fazer upload de rastreadores, que são imagens que podem ser usadas para amarrar os modelos na cena caso encontradas pela câmera. O aplicativo móvel do Augment é a parte principal do processo. Ele é sincronizado com o repositório online, disponibilizando assim todos os modelos importados. Ao selecionar um modelo da sua galeria ou de alguma outra que esteja como pública, a câmera do dispositivo é aberta posicionando inicialmente o objeto no centro. É possível mover o modelo usando um dedo na tela, para alterar o tamanho fazendo movimento de abertura e fechamento usando dois dedos e para rotacionar usar dois dedos fazendo o movimento vertical. Também existe a possibilidade de capturar um rastreador (Figura 2), ou seja, uma imagem na cena que determina a posição que o modelo deve ficar.

Figura 2- Renderização de realidade aumentada usando rastreador

Fonte: Augment (2018).

Quadro 3 – Pokémon GO: jogo baseado no desenho Pokémon

Referência	Prado (2016)

Objetivos	Procurar e capturar Pokémons espalhados pelo mapa.
Principais funcionalidades	Localizar Pokémons espalhados pelo mapa utilizando GPS para identificar e então visualizá-los utilizando realidade aumentada.
Ferramentas de desenvolvimento	Não foi encontrado.
Resultados e conclusões	Dentro do proposto, o jogo realiza bem sua funções já que ele não exige precisão ao posicionar os Pokémon dadas as coordenadas. Comercialmente falando, o produto obteve bastante sucesso, chegando à receita de 3 bilhões de dólares (SILVA, 2019).

Fonte: elaborado pelo autor.

O jogo funciona basicamente com um mapa que também é a tela principal do aplicativo. Nesse mapa é possível localizar Pokémons ao redor da posição atual do usuário. Conforme o usuário se locomove, novos Pokémons vão aparecendo. Ao selecionar um, a câmera abre mostrando Pokémon selecionado, permitindo que ele seja visto de duas maneiras: a primeira é uma animação em 3D em um cenário totalmente virtual (processo que não sobrecarrega tanto o dispositivo) e a outra é renderizar o Pokémon com RA. Os Pokémons encontrados no mapa não aparecem só para um usuário, eles são vinculados com coordenadas e qualquer pessoa que passar perto daquela localização consegue ver o Non-Player Character (NPC) também.

Após entrar no modo visualização (Figura 3), o último passo é tentar capturá-lo. Para isso, o usuário deve usar suas Pokébolas e fazer um movimento vertical no *touch* para lançar ela sobre o Pokémon. O percentual de sucesso da captura é baseado no nível do Pokémon.

Figura 3 – Pokémon com realidade aumentada

Fonte: Prado (2016).

Quadro 4 – Snapchat: rede social para compartilhar fotos e vídeos

Referência	Hamman (2016)
Objetivos	Receber e compartilhar fotos e vídeos com filtros que utilizam realidade aumentada.
Principais funcionalidades	Tirar fotos com diversos filtros que utilizam realidade aumentada através de rastreamento de marcadores (rosto do usuário).
Ferramentas de desenvolvimento	Não foi encontrado.
Resultados e conclusões	O aplicativo cumpre com o proposto e é uma das redes sociais mais utilizadas no mundo, segundo SimilarWeb (2019), plataforma que realiza análise de sites e aplicativos.

Fonte: elaborado pelo autor.

Ao abrir o aplicativo, a tela principal é a câmera, nessa tela é possível tirar uma foto clicando no botão de captura ou gravar vídeo mantendo o botão pressionado. O aplicativo permite que o usuário mantenha uma lista de contatos, permitindo que suas fotos e vídeos possam ser compartilhados com os mesmo ou postados de forma pública para qualquer pessoa.

O modo de edição do Snapchat ocorre em dois momentos. No primeiro é possível adicionar um filtro ou figura sobre a foto ou vídeo enquanto é feito a captura ou realizar a aplicação após esse processo. Os filtros disponíveis são basicamente alteração de cor e sobreposição de outras imagens disponibilizadas pelo próprio aplicativo.

3 DESCRIÇÃO DA BIBLIOTECA

Esta seção tem como objetivo apresentar uma visão geral do funcionamento e aspectos relevantes em relação à construção da biblioteca. Para tanto, são apresentadas quatro subseções. A primeira apresenta brevemente o funcionamento e possibilidades de implementação da biblioteca. Na segunda e terceira subseção são apresentadas as duas camadas que compõe o projeto. Por último, é apresentada uma aplicação de teste que foi implementada utilizando a biblioteca.

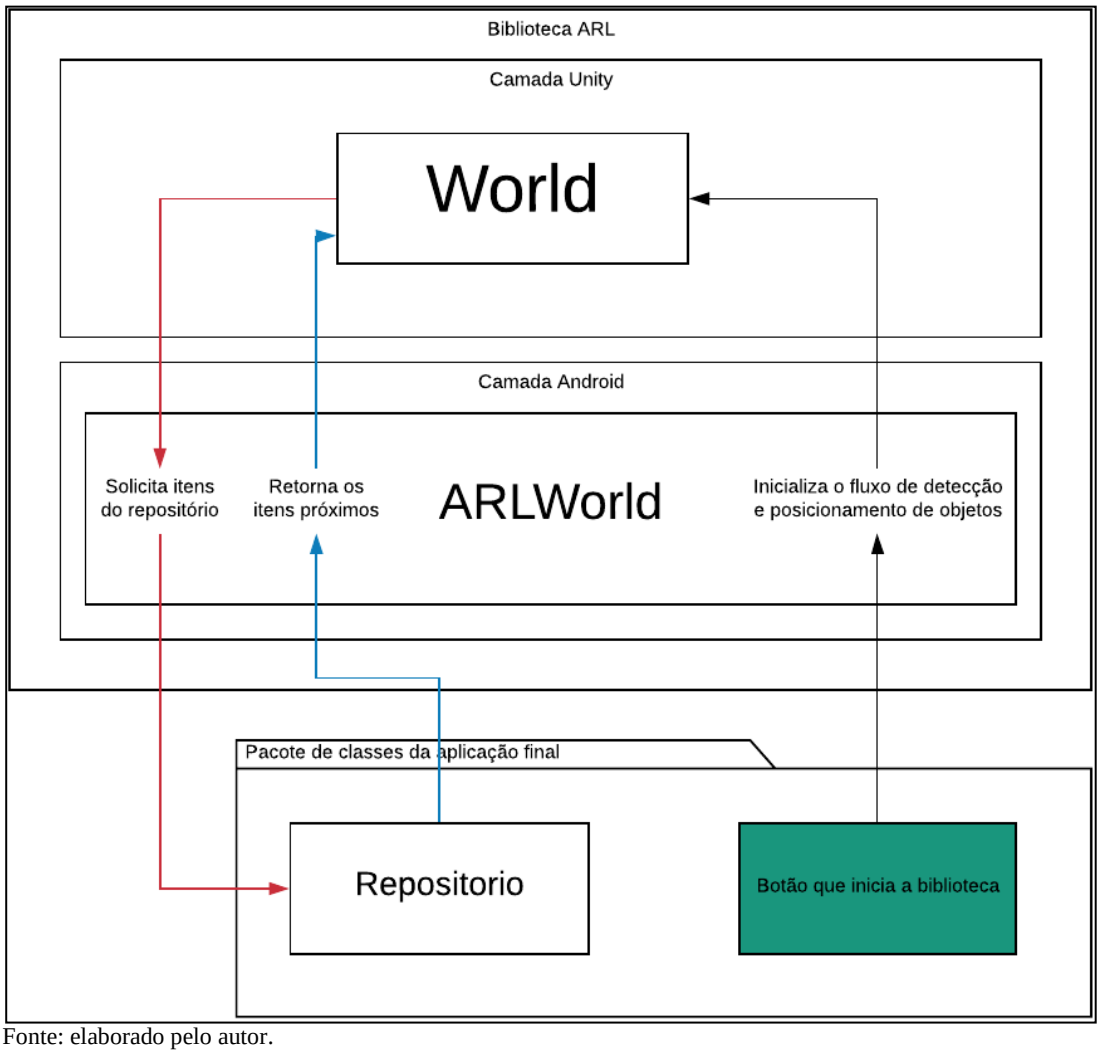
3.1 VISAO GERAL DA BIBLIOTECA

A biblioteca tem como principal objetivo permitir que objetos sejam visualizados utilizando realidade aumentada em qualquer lugar do mundo, já que faz uso de coordenadas geográficas. Ela disponibiliza ao desenvolvedor várias possibilidades de implementação, permitindo que seja possível compartilhar tanto seus modelos construídos, quanto modelos importados por seus usuários, tudo dependendo da forma que a aplicação for construída. Para que isso seja possível, a biblioteca conta com a abstração do repositório a ser utilizado, deixando para desenvolvedor definir se esse será em memória, na nuvem, em um banco de dados local ou qualquer outra forma.

Os modelos a serem posicionados em cena, por questões de performance e de uso de memória, não são todos buscados de uma vez só. O desenvolvedor tem liberdade para definir uma distância mínima para que eles sejam detectados. Então, conforme o usuário se move, novos objetos são renderizados e os que forem se distanciando são destruídos. O processo de posicionamento é feito automaticamente a partir das coordenadas do dispositivo do usuário e as cadastradas para cada modelo.

O projeto é dividido em duas camadas: a primeira foi desenvolvida com a ferramenta Unity utilizando a linguagem `C#` e a biblioteca `ARCore`. Essa camada é responsável por identificar a posição geográfica do dispositivo através de seus sensores e buscar os objetos cadastrados próximos, os posicionar resolvendo alguns algoritmos para que sejam visualizados utilizando realidade aumentada. A outra camada funciona como uma ponte entre o desenvolvedor e o funcionamento da aplicação. É nessa parte em que toda a regra de negócio do projeto é encapsulada em forma de biblioteca para que outras aplicações possam utilizar. Ela foi desenvolvida em Java com a ferramenta Android Studio. Na Figura 4 é apresentada uma visão geral das camadas.

Figura 4 – Visão geral da estrutura da biblioteca e interações com a aplicação final

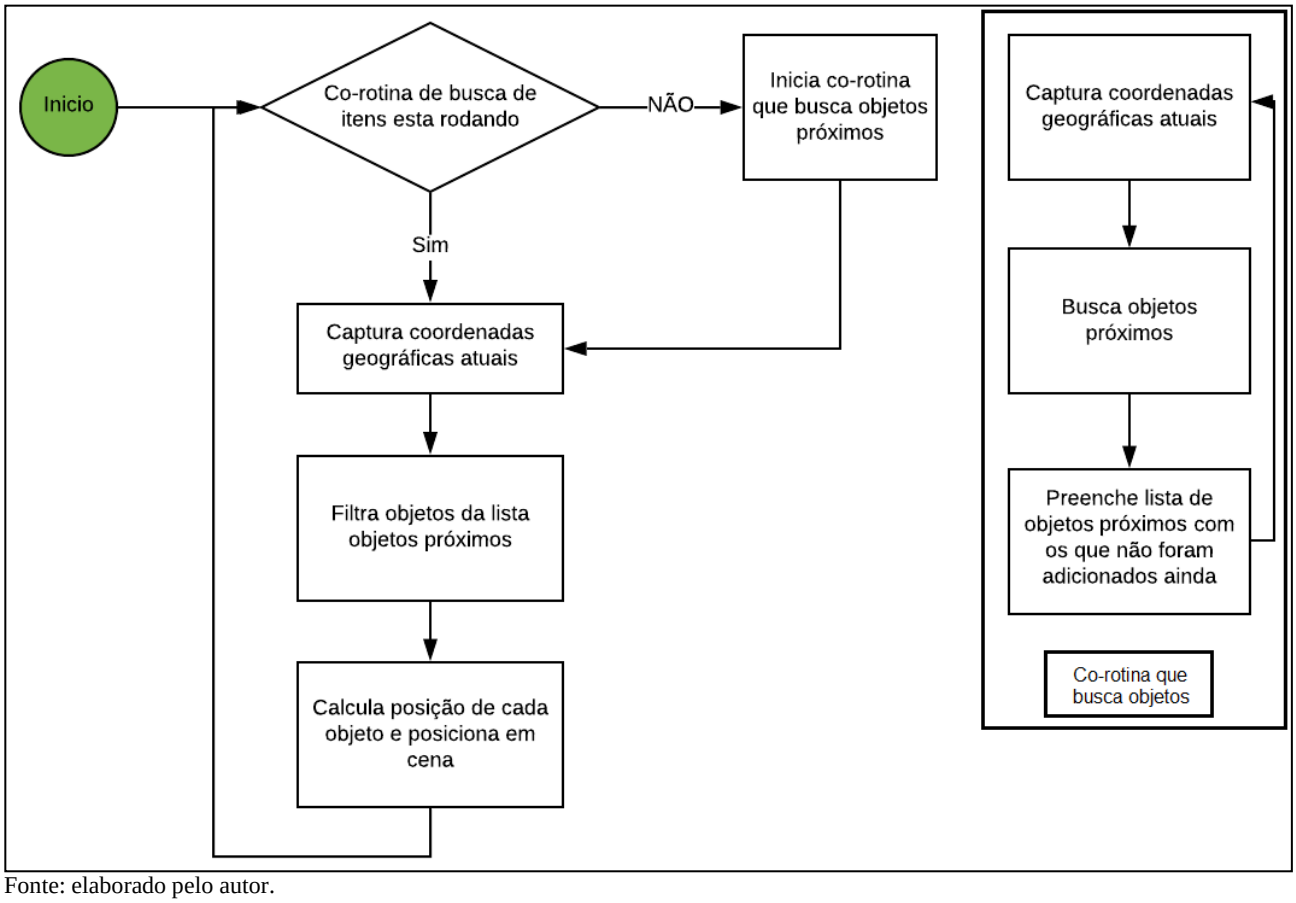


Fonte: elaborado pelo autor.

3.2 PRIMEIRA CAMADA (UNITY)

Esta camada contém as regras de negócio da biblioteca. Toda a parte de código que lida desde identificar a posição atual do usuário e buscar objetos próximos até o posicionamento e renderização deles é feito nela (Figura 5). Toda a execução dessa camada é gerenciada pela classe `UnityPlayerActivity` gerada ao exportar o projeto para o Android Studio. Enquanto executando, essa camada realiza seu fluxo em *loop*, até que seja interrompido ou pausado.

Figura 5 – Fluxograma de busca e posicionamento dos objetos próximos



Fonte: elaborado pelo autor.

Para a construção dessa camada foram utilizados componentes que são chamados de `GameObject` (termo utilizado para objetos gráficos ou não, do Unity). Cada `GameObject` pode ter um ou mais scripts que executam um conjunto de comandos a cada `frame` da cena. A cena é a representação virtual do mundo real que carrega e controla o ciclo de vida de todos os componentes que a compõe.

Conforme o fluxograma da Figura 5, após o início da execução, o script do `GameObject World` executa dentro do seu método `Update` (método padrão dos scripts de `GameObjects` executado a cada `frame` da cena) o código principal e se comunica com todas as outras partes para obter as informações necessárias.

O fluxo de posicionamento de objetos inicia com consulta das últimas coordenadas geográficas obtidas do dispositivo. As coordenadas são compostas de latitude e longitude e estão armazenadas no `script DeviceScript` (Quadro 5) que pertence ao `GameObject Device`. As coordenadas geográficas do dispositivo são atualizadas a cada um segundo, enquanto o ângulo do norte magnético é atualizado a cada frame. Essa diferença nos tempos de atualização foi definida, pois o GPS tem um tempo de resposta maior e pode acabar causando lentidão nas outras rotinas.

Quadro 5 – Código responsável por atualizar dados do GPS e bússola.

```
20      void Start()
21      {
22          Inicialização e verificação de permissões
35
36          var locationService = UnityEngine.Input.location;
37          locationService.Start(10, 0.01f);
38
39          new Thread(new ThreadStart(() =>
40          {
41              while (true)
42              {
43                  if (locationService.isEnabledByUser)
44                  {
45                      lastLocationInformation = locationService.lastData;
46                      localInfoReady = true;
47                  }
48
49                  Thread.Sleep(1000);
50              }
51          })).Start();
52      }
53
54      void Update()
55      {
56          lastNorth = compassService.magneticHeading;
57
58          if (mainCamera.transform.rotation.eulerAngles.x > 180)
59          {
60              lastNorth += 180;
61
62              if (lastNorth > 360)
63                  lastNorth -= 360;
64          }
65
66          Calibragem
83      }
```

Fonte: elaborado pelo autor.

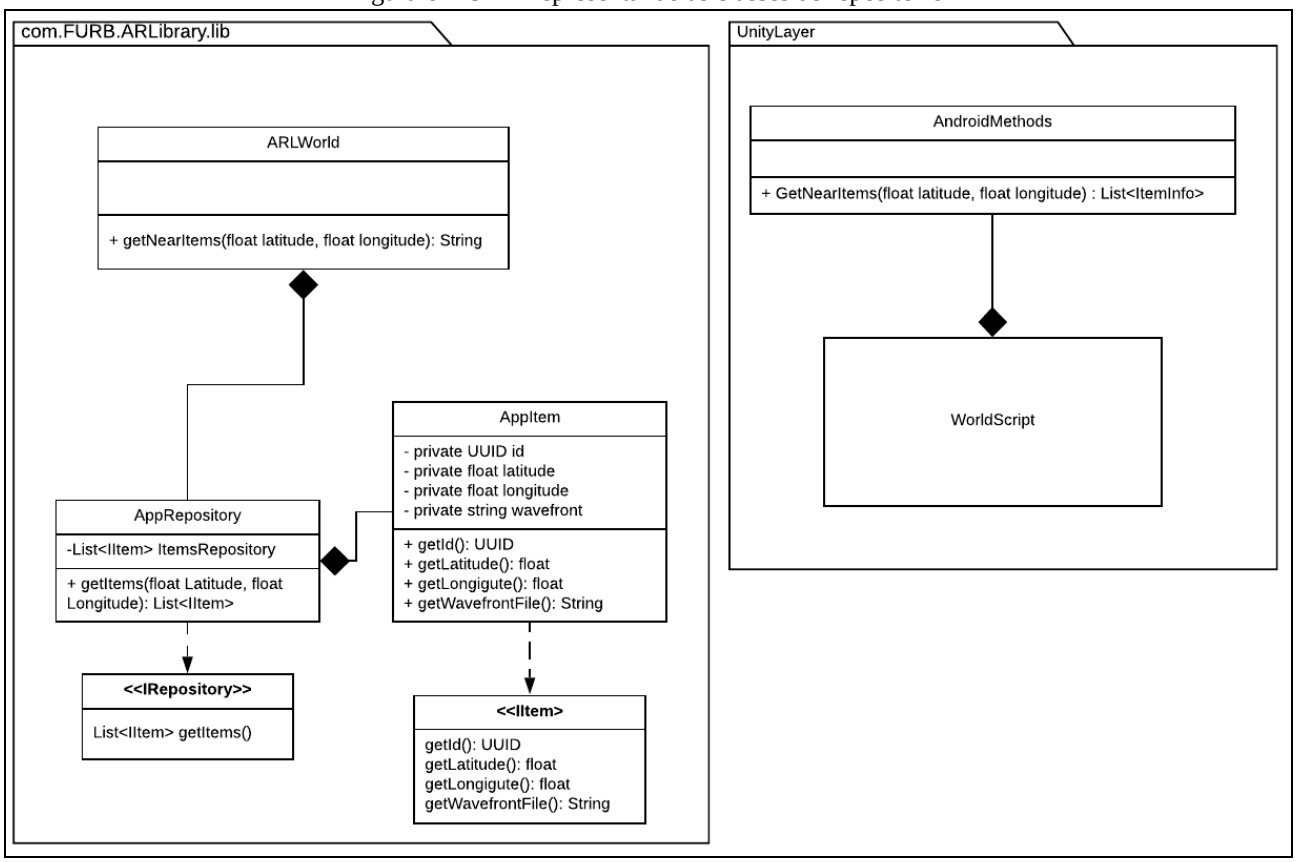
Com as coordenadas geográficas atuais do usuário, é possível identificar quais objetos foram cadastrados próximos a ele. Para isso, é realizado um filtro da lista de objetos próximos (`NearItems`) armazenados. Essa lista é alimentada por uma co-rotina que executa a cada cinco segundos buscando modelos próximos com as últimas coordenadas registradas no `DeviceScript`.

A busca dos objetos não é feita sob demanda na `thread` principal, e sim através de uma co-rotina. Isso ocorre por dois motivos. O primeiro é para que seja possível controlar os objetos já posicionados na cena. Todos os objetos rastreados são armazenados numa lista que não permite itens duplicados (diferencia cada um por seu identificador) e mantém um estado para cada, identificando o que foi ou não posicionado. O segundo motivo é performance. Acessar o repositório por demanda pode acabar trazendo lentidão dependendo da forma que o desenvolvedor o implementou.

A consulta realizada a cada cinco segundos para preencher a lista `NearItems` é uma chamada feita para a classe `AndroidMethods`, que é a ponte de comunicação entre as duas camadas da biblioteca. Essa comunicação é feita utilizando as classes `AndroidJavaClass` (que representa uma classe Java) e `AndroidJavaObject` (que representa a instância de um objeto instanciado a partir de uma classe em Java). A classe `AndroidJavaClass` é instanciada para a classe `ARLWorld` que pertence a segunda camada e é responsável por instanciar toda a aplicação. Sua instância é armazenada em um `AndroidJavaObject`.

Após obter a instância da classe `ARLWorld` é chamado seu método `getNearItems`. Nele são passados os parâmetros de latitude e longitude atuais para que o desenvolvedor possa filtrar as informações devolvidas e não precise carregar todos os objetos, assim evitando problemas de memória e performance. Na Figura 6 é possível visualizar a estrutura das classes de repositório das duas camadas e sua integração.

Figura 6 – UML representando as classes de repositório



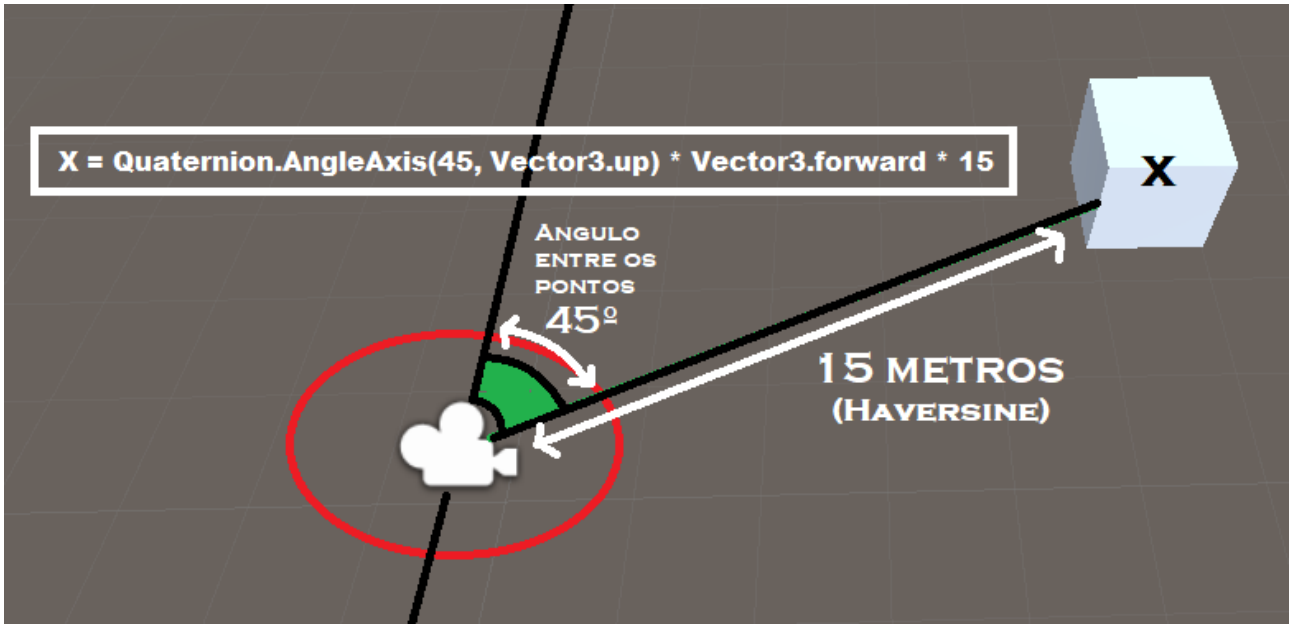
Fonte: elaborado pelo autor.

Todas as informações dos objetos rastreados são transportadas no formato de `JSON` e são deserializados (processo de construir um objeto) utilizando a classe `ObjectInfo`. Cada item retornado possui as seguintes informações: o `id` do objeto, as coordenadas geográficas e o modelo que será renderizado. O `id`, como já citado, é utilizado para controlar o que foi ou não detectado e renderizado. As coordenadas servem para posicionar o objeto em relação ao usuário e o modelo, que vem no formato `Wavefront` (`obj`), serve para criar uma `mesh` que é anexada ao objeto posicionado na cena. A `mesh` é um componente nativo do Unity e é a estrutura visual dos `GameObjects`. Objetos que não possuem `mesh` ficam invisíveis. O processo de conversão do arquivo `obj` é realizado pela classe `ObjConverter` que é chamada pelo método `Build` da classe `ObjectInfo` antes do `GameObject` ser posicionado na cena.

Após obter a lista de objetos rastreados, com todas as informações necessárias (`id`, `latitude`, `longitude` e `mesh`), são feitos dois cálculos para cada item: um para determinar a que distância o objeto deve ficar do dispositivo e outro para determinar em qual direção. Para calcular a distância se utiliza o algoritmo de Haversine, implementado pelo método `HaversineDistance`. Já para calcular o ângulo entre os pontos é utilizado azimute, implementado pelo método `AngleBetween`. Todos os algoritmos utilizados estão localizados no script `Functions` e recebem como parâmetros as coordenadas atuais do dispositivo e as coordenadas do objeto.

Para saber exatamente a posição do objeto a ser posicionado, é multiplicado o ângulo do objeto por um vetor *forward*, que define a direção do ângulo no eixo Z, e por fim multiplicando pelo valor da distância, e assim obtendo o vetor da posição em que o novo objeto deve ficar. Na Figura 7 é demonstrado como o cálculo funciona.

Figura 7 – Demonstração do cálculo realizado para posicionar novos objetos

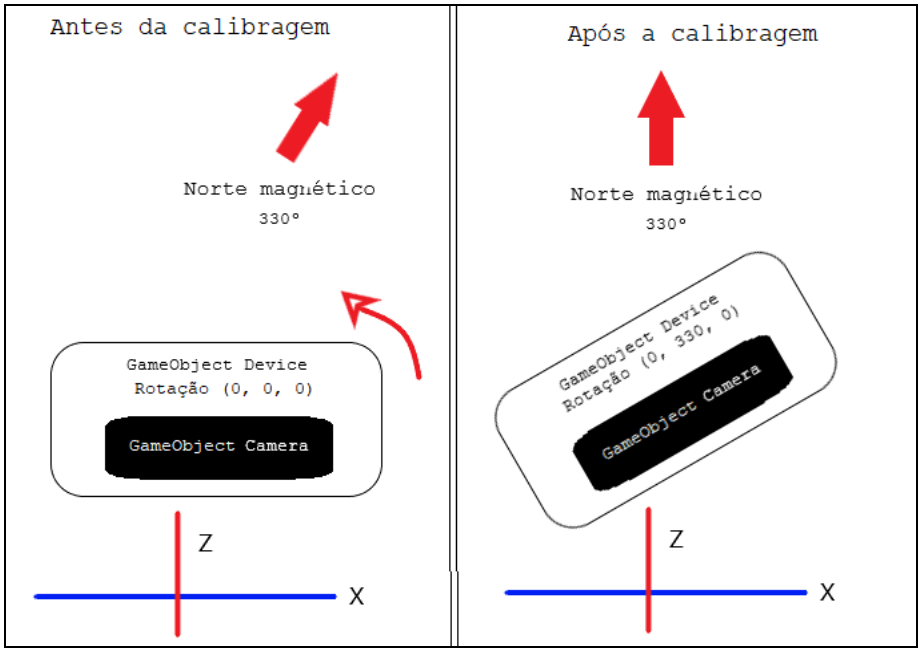


Fonte: elaborado pelo autor.

Com a posição e direção calculada, é instanciado para cada objeto da classe `ObjectInfo` um novo `GameObject` baseado no `Prefab DefaultObjectContainer`. Esse `GameObject` possui a `mesh` construída e um script. Nesse script são executados alguns comandos, entre eles é definido a rotina responsável por destruir objetos que saiam da distância mínima do dispositivo, assim não alocando memória em excesso.

Dentro da cena, além do `WorldScript`, existe a câmera que está localizada dentro do `Device`. A câmera possui a função de renderizar a imagem capturada pela câmera do dispositivo e simular a movimentação e rotação dele. Quando o aplicativo é iniciado, ela sempre tem a posição e rotação inicial zeradas e por isso é realizado um processo de calibragem em relação ao norte magnético ao iniciar a aplicação. Para realizar a calibragem é rotacionado o `GameObject Device` (Figura 8), assim a câmera inicia apontada para a mesma direção que usuário no mundo real, descartando a necessidade de consultar constantemente a bússola do aparelho para posicionar os objetos encontrados. Após isso, a rotação da câmera só é atualizada com o giroscópio.

Figura 8 – Processo de calibragem em relação ao norte



Fonte: elaborado pelo autor.

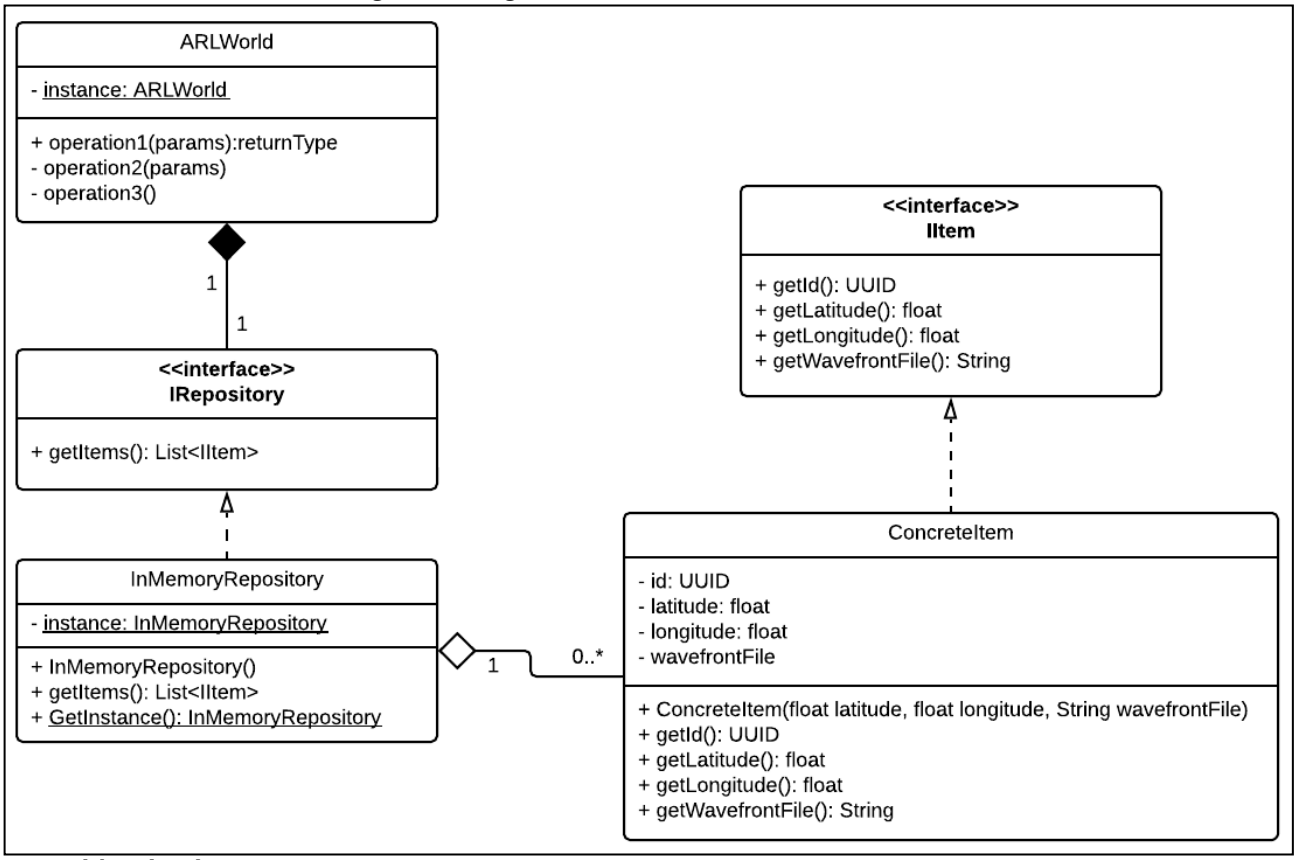
Junto ao `GameObject` da câmera, há alguns componentes anexados a ela, responsáveis por suas funções de simular o movimento do dispositivo e renderizar a imagem capturada. Para capturar os movimentos de deslocamento e

rotação é utilizado o `Tracked Pose Driver`. Esse componente é nativo do Unity e tem a função de buscar dados do giroscópio e acelerômetro para rotacionar e movimentar o objeto virtual junto com o dispositivo físico. E para renderizar a captura da câmera é utilizado o `AR Core Background Renderer`, um componente da biblioteca do ARCore que captura a imagem da câmera do dispositivo e renderizar como fundo da cena.

3.3 SEGUNDA CAMADA (ANDROID)

A segunda camada, utilizada na biblioteca, foi gerada a partir da exportação da primeira adicionando algumas classes para encapsular as chamadas da cena do Unity. Foram também adicionadas interfaces para permitir que o desenvolvedor implemente onde serão alocados os objetos a serem renderizados na aplicação (na nuvem, arquivo de texto, banco local, ou outro). A estrutura das classes foi definida conforme a Figura 9.

Figura 9 – Diagrama de classes da camada da biblioteca



Fonte: elaborado pelo autor.

A classe principal da aplicação é a `ARLWorld` que recebe três parâmetros: o primeiro é um contexto, geralmente da `Activity` fazendo a chamada, o segundo é um botão que será utilizado para iniciar a aplicação; e por último uma classe qualquer que deve, obrigatoriamente, implementar a interface `IRepository`. O primeiro parâmetro passado é necessário para permitir que sejam chamadas outras atividades de dentro da classe `ARLWorld` com o método `StartActivity`, nativo do Android. O botão passado como segundo parâmetro é configurado no construtor da aplicação, definindo um evento de `OnClick` para instanciar o *core* da aplicação (classe `UnityPlayerActivity`, gerada na exportação do projeto Unity) como se fosse uma `Activity` comum do Android (classe que geralmente representa uma tela em aplicações Android).

Por último, a classe de repositório é uma classe escrita pelo desenvolvedor final e que deve, obrigatoriamente, implementar a interface `IRepository` (Quadro 5) e o método `getItems`, que como já descrito na seção da primeira camada, será chamado para obter os objetos a serem renderizados na aplicação. O método `getItems` disponibiliza dois parâmetros de latitude e longitude do dispositivo, permitindo que o desenvolvedor possa filtrar o que será retornado. Essa solução é necessária para que, nos casos onde o repositório busque de um banco de dados ou da nuvem, os objetos possam ser filtrados antes de serem consultados, assim melhorando a performance da busca.

Para garantir que o desenvolver retorne todas as informações necessárias, fica definido que o método `getItems` deve retornar objetos que implementam a interface `IItem`, que obriga que métodos `GetId`, `GetLatitude`, `GetLongitude` e `GetWavefront` sejam definidos. A comunicação com a primeira camada é feita através da classe

`ARLWorld` através do método `getNearItems` que lê o método `getItems` do repositório e serializa os itens no formato `JSON` (Quadro 6).

Quadro 6 – Método que se comunica com a primeira camada da biblioteca

```
51      public String getNearItems(float latitude, float longitude){
52          String retorno = "";
53          List<IItem> itemList = repository.getItems(latitude, longitude);
54
55          for (IItem item : itemList) {
56              if (!retorno.isEmpty())
57                  retorno += ",";
58
59              retorno += String.format("{ \"Id\": \"%s\", \"Latitude\": %s, \"Longitude\": %s, \"Wavefront\": \"%s\", \"Placed\": false }",
60                      item.getId().toString(), item.getLatitude(), item.getLongigute(),
61                      item.getWavefrontFile().replace( target: "\r", replacement: "").replace( target: "\n", replacement: "\\n"));
62          }
63
64          return String.format("{ \"Items\": [%s] }", retorno);
65      }
```

Fonte: elaborado pelo autor.

3.4 APLICAÇÃO DE TESTES

Durante o desenvolvimento do trabalho, foi criado uma aplicação de testes e foram feitas algumas alterações para facilitar as validações. A aplicação de teste criada foi construída seguindo os passos disponíveis na documentação da biblioteca (SILVA, 2019), sendo possível assim, testar sua usabilidade. Ela é composta por um menu inicial que da acesso para o cadastro e consulta de itens e possui um botão para iniciar a aplicação que é passado no construtor da classe `ARLWorld`. No Quadro 7 é apresentado como a aplicação de teste implementou a classe de repositório, um exemplo básico de sua implementação em memória.

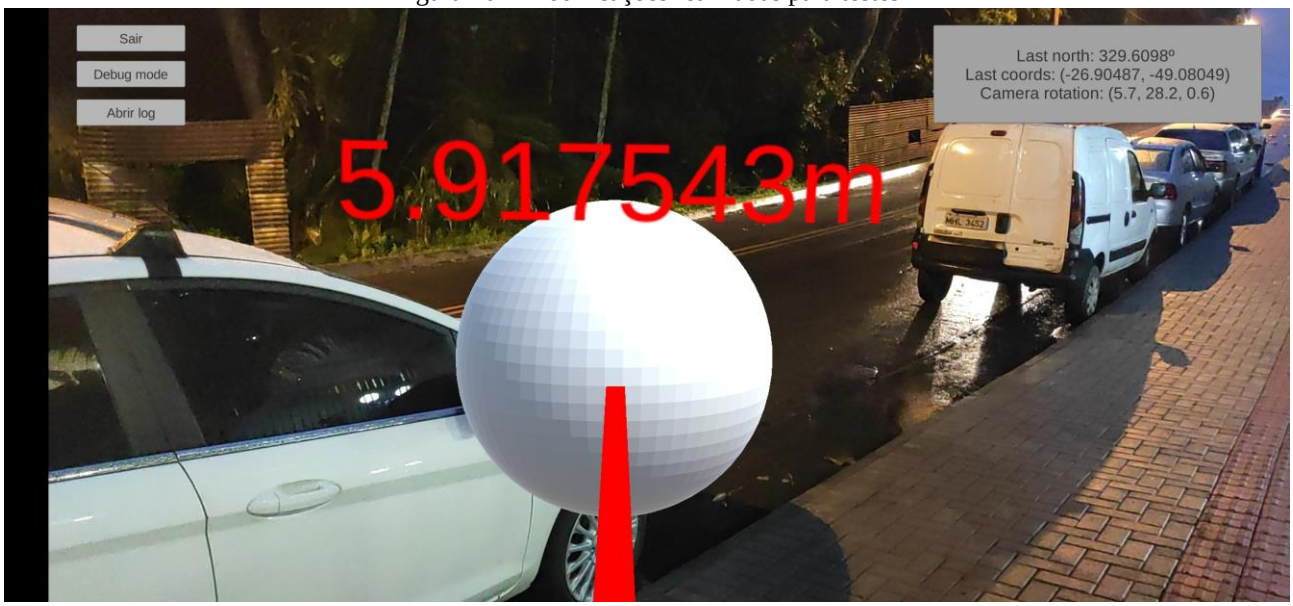
Quadro 7 – Implementação em memória da classe de repositório

```
7   public class ItemsRepository implements IRepository {
8       private static ItemsRepository instance;
9       private List<IItem> itemList;
10
11      public ItemsRepository(){
12          instance = this;
13          this.itemList = new ArrayList();
14      }
15
16      public List<IItem> getItems()    {
17          return itemList;
18      }
19
20      public List<IItem> getItems(float latitude, float longitude) { return itemList; }
23
24      public static ItemsRepository GetInstance() { return instance; }
27
28  }
```

Fonte: elaborado pelo autor.

Durante o desenvolvimento, a aplicação de testes sofreu algumas modificações para auxiliar o desenvolvimento e validação da biblioteca. Para facilitar encontrar os objetos na cena foi adicionado uma linha entre o usuário e o objeto posicionado e para validar a distância em que cada modelo foi ancorado, foi adicionado um texto em cima deles registrando a informação. Foram também adicionados na tela alguns logs de execução, informações da bússola e posição da câmera para verificar a eficiência da calibragem. Na Figura 10 são apresentadas as modificações implementadas.

Figura 10 – Modificações realizadas para testes



Fonte: elaborado pelo autor.

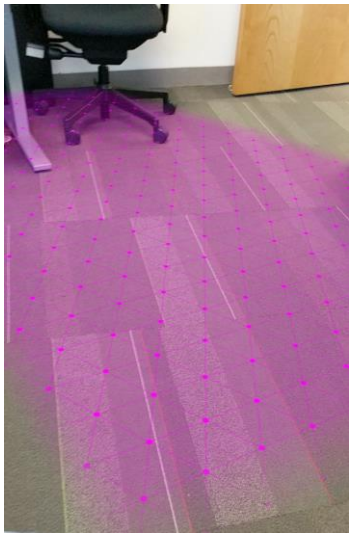
4 RESULTADOS

A seção de resultados foi dívida em três partes. Na primeira são apresentados os testes feitos em relação à biblioteca e componentes. Na segunda são descritos os testes feitos com os sensores. E por fim são apresentadas as abordagens feitas na construção da biblioteca.

4.1 TESTES DE BIBLIOTECAS

O primeiro passo na construção do projeto foi construir uma aplicação de testes no Unity com o intuito de testar a biblioteca `ARCore`. Essa biblioteca, desenvolvida pela Google, fornece diversas funções para construir aplicações utilizando realidade aumentada. Inicialmente, o objetivo era utilizar a função para detecção de planos horizontais (Figura 11) para posicionar os objetos cadastrados na cena, porém a ideia foi descartada, pois o processo de detecção de planos além de lento, era instável, detectando alguns planos e outros não.

Figura 11 – Função de detecção de planos do ARCore



Fonte: Codelabs (2018).

Após descartar a função de detecção de planos horizontais, foram realizados testes com a detecção de planos verticais. O objetivo era simular as paredes detectadas na cena para que, quando o usuário estivesse em ambientes internos, os objetos de fora fossem ocultados atrás dessas paredes virtuais. Essa ideia também não foi continuada pelo mesmo problema da primeira, a detecção era lenta e instável e acabou sendo pouco eficiente, já que as paredes eram apenas renderizadas nos locais para qual a câmera era apontada. Por fim, foram utilizados algumas classes e componentes do `ARCore` para auxiliar na captura de vídeo e detecção do giroscópio e acelerômetro, entre eles:

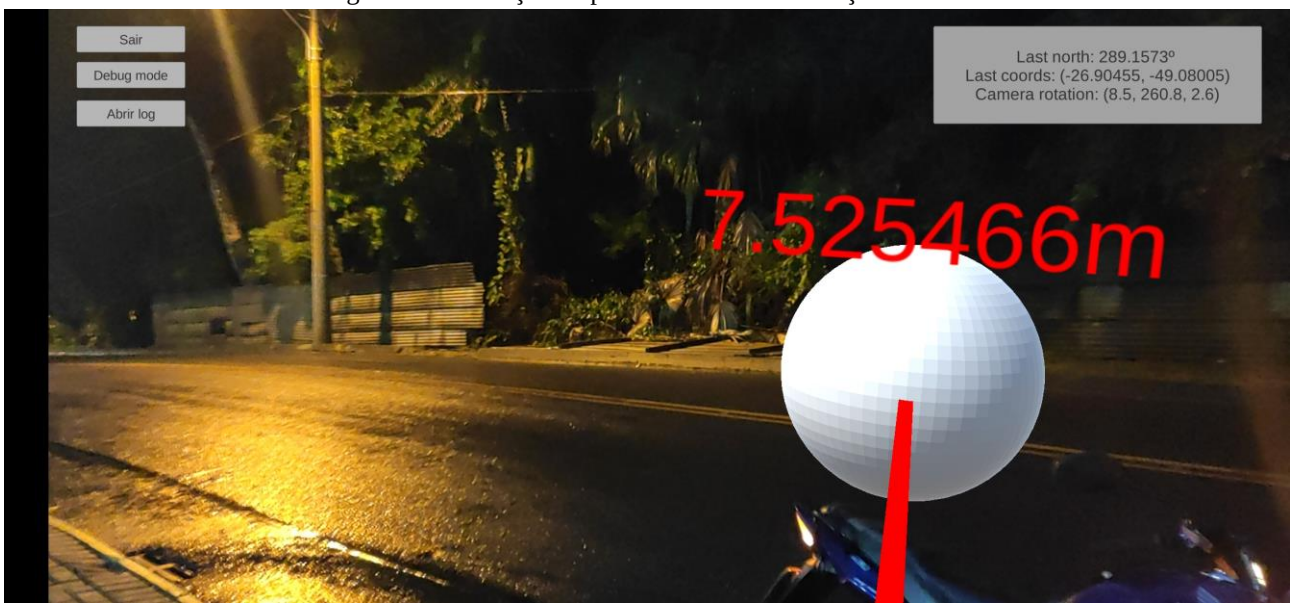
d) `AR Core Session`: utilizado definir alguns parâmetros de funcionamento, como: com qual câmera será realizado a captura de vídeo, qual a forma de detecção de planos irá ser utilizada, configurações de luz e configuração de foco (essas foram as utilizadas);
e) `AR Core Background Renderer`: anexado a câmera, esse componente define como fundo a captura de vídeo feita pela câmera.

O principal motivo para a utilização da biblioteca `ARCore` foi para a captura de vídeo, pois como o vídeo é renderizado como fundo, não ocorre o problema de sobrepor os `GameObjects` da cena. Ao tentar desenvolver a renderização de vídeo da câmera utilizando as bibliotecas do Unity, houve diversos problemas, pois seria necessário adicionar o vídeo como textura em um painel 2D e assim acabava cobrindo alguns objetos dependendo de sua posição.

4.2 TESTES DE SENSORES

Além dos testes realizados com a biblioteca `ARCore` nessa fase inicial, foram também testados alguns sensores para medir o nível de precisão e estabilidade de cada um. Dos sensores testados, foram constatadas algumas limitações de uso. O GPS, que é utilizado para detecção e posicionamento de objetos, apresentou uma variação de dez metros na sua posição e instabilidade em algumas áreas. A falta de precisão do GPS tem maior impacto visual em objetos menores, na Figura 12 foi cadastrado um objeto na latitude -26,90444 e longitude -49,07989, coordenadas capturadas em cima de calçada e ao iniciar o posicionamento e renderização, o objeto apresentou uma diferença de aproximadamente 7,5 metros, sendo posicionado na rua.

Figura 12 – Variação no posicionamento em relação ao GPS

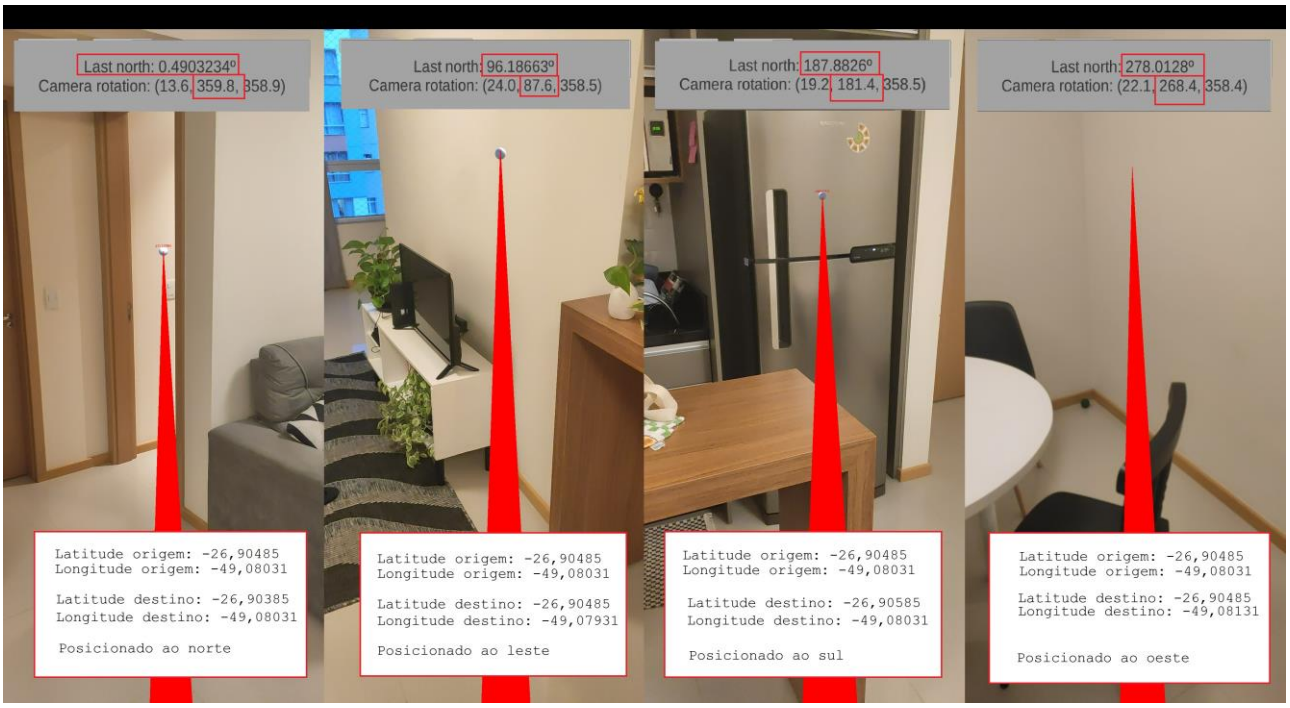


Fonte: elaborado pelo autor.

Os testes realizados em cima da bússola foram bastante satisfatórios e foi o sensor que mostrou o melhor desempenho. Apesar de ter uma alta sensibilidade em relação aos movimentos feitos, essa questão não foi um problema já que a informação do norte magnético é utilizada somente na inicialização da aplicação para calibrar o mundo virtual do Unity com a posição do dispositivo. Toda a rotação feita com o dispositivo a partir dali é feita pelo giroscópio. Durante os testes com a bússola, foi notado que ao girar o dispositivo no eixo X apontando para cima, o norte é invertido, como se o celular estivesse virado de costas, por isso foi aplicado uma rotina para que toda vez que o celular é inclinado para essa posição, o norte seja invertido.

Para validar os testes com a bússola, foram feitos quatro testes de mesa posicionando objetos em cada direção: norte, leste, sul e oeste, respectivamente, conforme a Figura 13. Na figura são apresentados dados de origem e destino de latitude e longitude, dados sobre o norte magnético e a posição da câmera, destacando o eixo Y, que através do processo de calibragem, deve estar o mais próximo do ângulo do norte. Para posicionar os objetos, foi capturada a coordenada atual (latitude e longitude) e alterado seu valor. No primeiro, que posicionado ao norte, foi somado 0,001 à latitude e o objeto ficou posicionado num ângulo bem próximo a 0º, ou seja, alinhado com o norte magnético. No segundo, foi adicionado 0,001 à longitude colocando o objeto à leste, bem próximo à 90º, apresentando uma diferença devido a calibragem não estar sincronizada corretamente. No terceiro foi subtraído 0,001 da latitude, colocando o objeto próximo a 180º, novamente demonstrando diferenças na calibragem. E por último, o objeto foi posicionado subtraindo 0,001 da longitude, assim, posicionando-o próximo à 270º, também havendo diferença na rotação da câmera.

Figura 13 – Testes feitos para avaliar a eficiência da bússola e giroscópio



Fonte: elaborado pelo autor.

Além do GPS e da bússola, foram realizados testes com giroscópio e acelerômetro. Os testes feitos foram utilizando o componente `Tracked Pose Driver` do Unity, que tem função de atualizar a rotação e posição de um objeto junto ao do dispositivo. O componente apresentou algumas instabilidades no posicionamento e rotação, por conta disso foi alterado para que somente atualizasse o posicionamento e a rotina que atualiza rotação foi implementada manualmente. Apesar da implementação manual do giroscópio ter funcionado melhor que a do componente individualmente, ao rodar as duas funcionalidades (rotação e translação) juntas houve problemas de sincronia entre as informações, e por isso foi mantida a ideia inicial de controlar rotação com o `Tracked Pose Driver`. Na Figura 13 é possível ver algumas diferenças na rotação da câmera em relação aos dados da bússola.

Levando em conta todos os testes realizados, foi previsto desde o início que os objetos não poderiam ser renderizados a distâncias muito longas, já que são posicionados em relação à bússola e um mínimo desvio de ângulo poderia crescer linearmente, resultando numa diferença muito grande no resultado.

4.3 CONSTRUÇÃO DA BIBLIOTECA

Inicialmente, o projeto seria construído somente utilizando o Unity, porém foi visto que não seria possível construir uma estrutura fácil de utilizar e que pudesse ser importada como uma biblioteca qualquer de Android. Após isso foi decidido gerar um projeto para Android Studio e a partir desse projeto, gerar um `Android Archive` (`aar`), formato de biblioteca do Android. Com essa biblioteca gerada, seriam implementadas as interfaces e classes para gerenciar a comunicação com a camada principal num novo projeto em Kotlin. Após apresentar várias falhas para gerar um versão estável, a estrutura da biblioteca foi simplificada, excluindo a camada implementada em Kotlin e implementando as alterações e interfaces diretamente no projeto exportado do Unity (em Java).

Na construção da biblioteca todos os objetivos formam alcançados. Foi construído uma estrutura intuitiva e que permite o usuário implementar seu próprio repositório e abstraindo qualquer necessidade de possuir conhecimento ou de implementar rotinas envolvendo realidade aumentada, uso dos sensores ou cálculos com coordenadas geográficas.

Para validar os testes de usabilidade da biblioteca, como citado na seção de descrição do projeto, foi desenvolvido uma aplicação para testes seguindo a documentação da biblioteca (SILVA, 2019). Após baixar as dependências e importar na aplicação, foi implementada a interface `IRepository` e `IItem` e criado telas de consulta e cadastro para preencher a lista em memória de dentro do repositório. Depois disso, na atividade principal da aplicação foi instanciado a classe `ARLWorld` passando como parâmetro o repositório e o botão que inicia a câmera (Quadro 8), sem necessidade de configurar mais nada.

Quadro 8 – Atividade principal da aplicação instanciando a biblioteca

```java
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);

        ItemsRepository repository = new ItemsRepository(); // repositório implementado

        // binding de botões
        Button btnCamera = (Button) findViewById(R.id.btnCamera);
        Button btnNewItem = (Button) findViewById(R.id.btnNewItem);
        Button btnGetItems = (Button) findViewById(R.id.btnGetItems);

        btnNewItem.setOnClickListener(v -> { // configurando tela de cadastro
            Intent itemsActivity = new Intent( packageContext: this, NewItemActivity.class);
            startActivity(itemsActivity);
        });

        btnGetItems.setOnClickListener(v -> { // configurando tela de consulta
            Intent itemsActivity = new Intent( packageContext: this, ItemsActivity.class);
            startActivity(itemsActivity);
        });

        ARLWorld arlWorld = new ARLWorld( activity: this, btnCamera, repository); // instanciando a aplicação
    }
}
```

Fonte: elaborado pelo autor.

5 CONCLUSÕES

Diante dos resultados apresentados conclui-se que, do ponto de vista do objetivo geral de construir uma biblioteca que permite compartilhar objetos 3D, ela cumpre com a sua proposta. Embora o posicionamento de objetos possa ser instável e, à longas distâncias, apresenta diferenças maiores, esse problema acaba por ser uma limitação dos sensores utilizados e que afeta não só nesse projeto, como outros trabalhos desenvolvidos que os utilizam.

Apesar dos problemas identificados na precisão dos sensores na etapa de posicionamento, os outros objetivos secundários foram cumpridos. Foram disponibilizadas formas de importar modelos próprios, limitados somente a arquivos `obj`. A parte de cadastro de objetos foi mesclada com o objetivo anterior, já que o repositório utilizado foi abstraído ao disponibilizar interfaces, assim deixando o cadastro de objetos para a definição do desenvolvedor. Para a extensão da biblioteca fica proposto:

f) implementar a possibilidade de importar outros formatos de arquivos;
g) permitir que sejam importados materiais e animações para os modelos;
h) adicionar a funcionalidade de posicionar os modelos em cena através da câmera para já poder visualizá-los na hora;
i) construir uma rotina para calibrar a rotação da câmera com o norte de forma periódica.

REFERÊNCIAS

ACZEL, Amir D.. **Bússola**: A invenção que mudou o mundo. 2001. Disponível em: <https://zahar.com.br/sites/default/files/arquivos/Trecho%20-%20B%C3%BAssola_0.pdf>. Acesso em: 17 dez. 2019.

ALECRIM, Emerson. **Snapchat usa realidade aumentada para colocar objetos 3D em fotos e vídeos.** 2016. Disponível em: <https://tecnoblog.net/213119/snapchat-world-lenses/>. Acesso em: 02 dez. 2019.

AUGMENT; **About Augment**. 2018. Disponível em: https://www.augment.com/about-us/. Acesso em: 28 set. 2018.

AZUMA, Ronald et al. Recent advances in augmented reality. **IEEE Computer Graphics And Applications**, [S.l.], v. 21, n. 6, p.34-47, 2001. Institute of Electrical and Electronics Engineers (IEEE).

CODELABS. **Introduction to ARCore in Unity.** 2018. Disponível em: <https://codelabs.developers.google.com/codelabs/arcore-intro/#5>. Acesso em: 05 dez. 2019.

FARIAS, Adelito et al. Educação em Saúde no Brasil: uma revisão sobre aprendizagem móvel e desafios na promoção de saúde no Brasil. **Anais**... [S.l.]: SBC, 2015, p.614-623.

HAMANN, Renan. **Snapchat:** tudo que você precisa saber para usar o app do momento. 2016. Disponível em: <https://www.tecmundo.com.br/redes-sociais/102784-snapchat-tudo-voce-precisa-saber-usar-app-momento.htm>. Acesso em: 02 dez. 2019.

KIRNER, Claudio; KIRNER, Tereza G. Development of an educational spatial game using an augmented reality authoring tool. **International Journal of Computer Information Systems and Industrial Management Applications**, v.3, p. 602-611, 2011.

LUTFI, Antonio N.; RAPOSO, Alberto. B. Realidade Aumentada e Publicidade: Até onde pode ir essa relação? In: Workshop de Realidade Virtual e Aumentada, 7., 2010, São Paulo. **Anais...**São Paulo: MACKENZIE, p. 113-118.

KUSTERS, Marc. **FastObjImporter.** 2015. Disponível em: <http://wiki.unity3d.com/index.php/FastObjImporter>. Acesso em: 02 dez. 2019.

MANSSOUR, Isabel H.; COHEN, Marcelo. Introdução à computação gráfica. **Revista de Informática Teórica e Aplicada**, Rio Grande do Sul, v. 13, n. 2, p. 1 - 25, 2006.

MCHENRY, Kenton; BAJCSY, Peter. **An Overview of 3D Data Content, File Formats and Viewers.** Urbana: Image Spatial Data Analysis Group, 2008. 21 p. Disponível em: <https://www.archives.gov/files/applied-research/ncsa/8-an-overview-of-3d-data-content-file-formats-and-viewers.pdf>. Acesso em: 02 dez. 2019.

PRADO, Jean. **Tudo sobre Pokémon Go**: um guia para você ser o melhor treinador. 2016. Disponível em: <https://tecnoblog.net/198133/pokemon-go-como-jogar/>. Acesso em: 02 dez. 2019.

PHILIPE, Gabriel. **Como funciona o GPS?** 2016. Disponível em: <https://www.oficinadanet.com.br/post/12406-como-funciona-o-gps>. Acesso em: 07 nov. 2018.

SETZER, Michael. **How compass apps can tell direction**. 2014. Disponível em: <https://thetartan.org/2014/1/27/scitech/compass>. Acesso em: 17 dez. 2019.

SILVA, Douglas Vieira da. **Pokémon GO ultrapassa US$ 3 bilhões em receita**. 2019. Disponível em: <https://www.tecmundo.com.br/mercado/147268-pokemon-go-ultrapassa-us-3-bilhoes-receita.htm>. Acesso em: 02 dez. 2019.

SILVA, Maicon Santos. **ARL Project**. Blumenau, 10 dez. 2019. Disponível em: https://github.com/maiconwazo/ARLProject. Acesso em: 05 dez. 2019.

SIMILARWEB. **Snapchat**: Ranking de uso. 2019. Disponível em: <https://www.similarweb.com/pt/app/google-play/com.snapchat.android/statistics#trafficSources>. Acesso em: 02 dez. 2019.

SOUZA, Wendson de Oliveira et al. A Realidade Aumentada na apresentação de produtos cartográficos. **Boletim de Ciências Geodésicas**, [S.l.], v. 22, n. 4, p.790-806, dez. 2016.