UNIVERSIDADE DE SÃO PAULO
Escola de Engenharia de São Carlos

Departamento de Engenharia Elétrica e Computação

Realidade Aumentada Aplicada em Decoração de Ambientes Desenvolvida para o Sistema Operacional Android

Lucas Carvalho Gomes

São Carlos - SP

Realidade Aumentada Aplicada em Decoração de Ambientes Desenvolvida para o Sistema Operacional Android

Lucas Carvalho Gomes

Orientador: Prof. Dr. Valdir Grassi Junior

Monografia referente ao projeto de conclusão de curso do Departamento de Engenharia Elétrica e Computação da Escola de Engenharia de São Carlos – EESC-USP para obtenção do título de Engenheiro de Computação.

Área de Concentração: Desenvolvimento *Mobile*, Realidade Aumentada

USP – São Carlos
31 de outubro de 2016

Gomes, Lucas Carvalho
G933r Realidade aumentada aplicada em decoração de ambientes desenvolvida para o sistema operacional Android / Lucas Carvalho Gomes; orientador Valdir Grassi Junior. São Carlos, 2016.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2016.

1. Realidade aumentada. 2. Android. 3. Desenvolvimento mobile. I. Título.

FOLHA DE APROVAÇÃO

Nome: Lucas Carvalho Gomes

Título: "Realidade aumentada aplicada em decoração de ambientes desenvolvida para o Sistema Operacional Android"

Trabalho de Conclusão de Curso defendido em 24 / 11 / 2016.

Comissão Julgadora:	**Resultado:**
Prof. Dr. Valdir Grassi Júnior (Orientador) - SEL/EESC/USP	Aprovado
Prof. Dr. Marcelo Andrade da Costa Vieira SEL/EESC/USP	APROVADO
Profa. Associada Kalinka Regina Lucas Jaquie Castelo Branco SSC/ICMC/USP	Aprovada

Coordenador do Curso Interunidades Engenharia de Computação pela EESC:

Prof. Dr. Maximiliam Luppe

Agradecimentos

Agradeço primeiramente a Deus, pela saúde concedida e sabedoria para realização deste trabalho.

À minha família pelo apoio e incentivo aos estudos e que mesmo longe sempre torceram pelo meu sucesso profissional.

Muito obrigado também à minha noiva e futura esposa que sempre esteve do meu lado e compartilhou comigo, mesmo distante, de toda minha trajetória.

Agradeço também aos meus amigos e colegas da universidade que sempre estiveram comigo e juntos pudemos vivenciar os momentos e estudar e partilhar do conhecimento obtido.

Ao meu orientador pela disposição e paciência para me auxiliar com as dúvidas que surgiram durante este trabalho, me dando todo suporte necessário.

E a todos os professores que desde o primário até aqui ajudaram em minha formação.

Resumo

A realidade aumentada é um conceito criado em 1981 que trata da sobreposição de objetos virtuais ao mundo real. Seu uso tem se tornado cada vez mais comum com o advento da miniaturização dos computadores e o seu consequente maior desempenho. De forma semelhante, os dispositivos móveis fazem parte atualmente do cotidiano de uma grande parcela da população, assim, a combinação destas duas tecnologias veio a facilitar a interação com a realidade aumentada. Este trabalho vem mostrar, então, as etapas de desenvolvimento de uma aplicação de realidade aumentada para Android e as ferramentas utilizadas no processo como o Vuforia, ferramenta de detecção de marcadores que dá apoio à realidade aumentada, e o libGDX que abstrai o OpenGL facilitando o processamento gráfico. Assim como as dificuldades de se produzir uma aplicação deste tipo que envolvem desde a descoberta de novas tecnologias até a aparição de requisitos não planejados. Por fim, o produto obtido por meio da metodologia ágil e da cuidadosa implementação se valendo de padrões de projetos e ferramentas de alto nível exibe fácil escalabilidade e rápida implementação se comparado a implementações feitas sem o uso das mesmas.

Sumário

Lista de Abreviaturas

API *Application Program Interface*

CAD *Computer Aided Design*

GPS *Global Positioning System*

HAL *Hardware Abstraction Layer*

HMD *Head Mounted Display*

JSON *JavaScript Object Notation*

NDK *Native Development Kit*

RA Realidade Aumentada

RV Realidade Virtual

SDK *Software Development Kit*

SO Sistema Operacional

UML *Unified Modeling Language*

Lista de Tabelas

Lista de Figuras

CAPÍTULO 1: INTRODUÇÃO

1.1. Contextualização e Motivação

A realidade aumentada (RA) é a sobreposição de um ambiente físico, do mundo real, com elementos virtuais gerados por computador por meio de estímulos sensoriais captados pelo mesmo, tais como som, vídeo, tato, força ou dados de GPS (*Global Positioning System*). Ela está relacionada com um conceito mais geral chamado realidade mediada, em que uma visão da realidade é modificada (possivelmente até mesmo diminuída, em vez de aumentada) por um computador. Assim, possui-se como resultado uma percepção multissensorial reforçada da realidade [10]. Em contraste, a realidade virtual (RV) substitui o mundo real com o simulado [28]. O aumento da realidade é convencionalmente em tempo real e dentro do contexto de seu ambiente, como a sobreposição de placares esportivos na TV durante um jogo. Com a ajuda de tecnologias de RA avançadas, por exemplo, a adição de visão computacional e reconhecimento de objetos, as informações sobre o mundo real em torno do usuário tornam-se interativas e manipuláveis digitalmente. Além disso, os sistemas de RA são implementados de forma que o ambiente real e os objetos virtuais permanecem ajustados, mesmo com a movimentação do usuário no ambiente real [6].

Apesar da RA estar mais relacionada a aplicações de representação visual, pode-se em principio aumentar qualquer interface da realidade, como é o caso do *Toozla* [36], uma espécie de guia turístico que se utiliza de GPS para comentar sobre a história do local em que o usuário se encontra. Alguns tipos de aplicações envolvendo RA são:

- Ramo automotivo e de navegação: é o caso de um pára-brisa desenvolvido pela *General Motors* (GENERAL MOTORS, 2011) em parceria com a *Carnegie Mellon University* e a *University of Southern California*, que utiliza a tecnologia HUD (*heads-up display*) para mostrar faixas, elementos de sinalização e animais mesmo em condições desfavoráveis [9];

- Prototipação de produtos: protótipos podem ser utilizados quando se deseja uma visão estrutural e estética do produto mais rapidamente, podendo-se agregar funcionalidades ao protótipo por meio de *software* e avaliá-lo de antemão. Este

é o caso do ARMO [15] (Figura 1), que reconhece objetos volumétricos e renderiza formas tridimensionais previamente desenvolvidas em programas de desenho auxiliado por computador (CAD);

- Entretenimento: a indústria do entretenimento investe alto em tecnologias inovadoras de RA e RV para se destacar. Um exemplo disso é o jogo *Eye Pet* desenvolvido para a plataforma de videogame Playstation 3 e mais recentemente o jogo *The Playroom* da plataforma de videogame Playstation 4 onde personagens virtuais interagem com pessoas e objetos do mundo real [32];

- Treinamento: treinar funcionários previamente utilizando-se RA antes de treiná-los com objetos reais é mais seguro, como é o caso do treinamento na indústria petrolífera realizado pelo projeto AMIRE [11];

- Consumidor final: aplicativos que apresentam facilidades para o cotidiano dominam esse ramo, por exemplo, ao se direcionar a câmera do dispositivo para determinado local, um programa é capaz de identificar pontos de referência e apresentar descrições sobre os mesmos (caso do *Nokia City Lens*) ou, então, ele detecta palavras em outro idioma e traduz para sua língua natal (*World Lens*);

Figura 1: Protótipo configurável para avaliação física de interfaces. Fonte: [11]

Assim, percebe-se a grande diversidade de produtos que se utilizam da tecnologia de RA. No entanto, não se encaixam no termo as aplicações de realidade virtual, edição fotográfica e de mídia aumentada. A última, apesar de ser bem parecida em termos práticos com a RA, não é considerada um ramo de RA por alguns autores. O fato de estender uma mídia e não o mundo a sua volta é o que desclassifica a mídia aumentada do termo. Um exemplo de mídia aumentada pode ser visto no *Guiness Book 2013*, que mostra uma

animação 3D dos recordes por meio de uma aplicação de celular quando se aponta para algumas páginas que contém *QR code* [27].

Uma aplicação interessante no ramo de realidade aumentada para dispositivos móveis é sobre decoração. Aplicações de decoração e arquitetura que permitem a visão 3D da mobília sobreposta à maquete do ambiente existem de longa data para computadores pessoais. Porém, essas aplicações não são tão bem exploradas no ramo de RA. Surge então uma motivação de complementar esses programas oferecendo uma oportunidade do usuário visualizar o resultado da decoração em um ambiente real.

Em relação aos aplicativos para dispositivos móveis de decoração que se utilizam de RA, pode-se dizer que a maioria falha ao não obedecer ao sentido de localidade fixa do objeto virtual ou de dependerem de *hardwares* com sensores específicos ou de alto desempenho. O destaque fica para o *IKEA catalogue* [37], o mais popular, que mostra os móveis do catálogo em RA e usa a revista como referência para o objeto virtual ser posicionado. Assim como para o *iStaging* [38] que não depende de um referencial pré-definido, porém sofre um pouco com flutuações ao mostrar o objeto e necessita de alguns poucos ajustes manuais.

Portanto, tendo em vista a fácil aderência por parte do consumidor a tecnologias de RA que facilitem o cotidiano, a crescente tendência desta tecnologia no mercado e as escassas aplicações disponíveis atualmente sobre decoração em AR, aliado ao interessante ramo de conhecimento dentro da computação gráfica e visão computacional em que o tema se encontra, surge a motivação deste trabalho.

1.2. Objetivos

O objetivo deste trabalho é desenvolver uma aplicação de realidade aumentada para dispositivos móveis por meio de metodologias ágeis e utilizando-se de padrões de projeto para bem estruturá-lo, de forma a se obter resultados visíveis em curtas iterações de projeto e do mesmo ser facilmente escalável.

Ademais, pretende-se apresentar conceitos de RA e ferramentas utilizadas na composição de projetos de RA de forma a apresentar as dificuldades e facilidades de cada uma e discutir sobre quais padrões são mais desejáveis neste tipo de projeto.

Por fim, o resultado final está pronto para o mercado, com *design* padronizado e desempenho otimizado para ser capaz de executar em uma gama de aparelhos diferentes, além disso, há interesse de que ele seja modular de forma que seja fácil atualizá-lo após o lançamento.

1.3. Organização do Trabalho

No capítulo 2 é apresentada uma revisão bibliográfica que servirá como base para o entendimento dos demais capítulos da monografia, apresentando os conceitos, metodologias, trabalhos de literatura na área e terminologia básica. A seguir, no capítulo 3 é apresentado o desenvolvimento do projeto, o qual apresenta cada etapa de planejamento seguindo as metodologias apresentadas no capítulo anterior, assim como os resultados e dificuldades obtidas. Finalmente, no capítulo 4 são apresentadas conclusões sobre o trabalho assim como os trabalhos futuros.

CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

2.1. Considerações Iniciais

A realidade aumentada tem suas bases em experimentos com realidade virtual, que por sua vez é relatada com a origem em experiências cinematográficas multimodais. Nesse sentido, destaca-se o Sensorama [39], de Morton Heilig, criado em 1956, o qual permitia um passeio pelas ruas de Manhattan com sensações de som, aromas, vibrações e vento sincronizados com o vídeo.

Já em 1961, os engenheiros da Philco Comeau, e Bryan, criaram um capacete HMD (*Head Mounted Diaplay*) cujos movimentos controlavam uma câmera remota, implementando-se assim a telepresença por vídeo [40]. Dois anos depois, Sutherland, apresentou em sua tese de doutorado um sistema que envolvia uma caneta óptica capaz de interagir com o computador para seleção e desenho de Figuras no monitor, intitulada de *Sketchpad, a Man-Machine Graphical Communication System* [29]. Nela ele estabeleceu termos que depois seriam utilizados em pesquisas de RV como gráficos no monitor, interação em tempo real e dispositivos especiais.

O mesmo Sutherland definiu, em 1965, conceitos de um *display* usado para interagir com objetos em um mundo virtual, envolvendo estímulos visuais, sonoros e táteis, de forma intuitiva em *The Ultimate Display* [30]. Finalmente, em 1968, Sutherland descreveu em seu artigo *A Head-Mounted Three Dimensional Display* [31] um capacete (HMD) estereoscópico e rastreável, que continha dois *mini-displays CRT* e uma interface de rastreadores mecânicos e ultra-sônicos, o que estabeleceu um marco histórico para realidade virtual em termos de imersão. Ambos os capacetes da Philco e de Sutherland estabeleceram as bases para realidade aumentada, que teve seus termos cunhados anos depois (1990) pelo Prof. Thomas Caudell, em uma visita à Boeing, em referência a um dispositivo que ajudava funcionários na montagem de equipamentos eletrônicos de aeronaves.

Apesar do termo ter aparecido apenas em 1990, a primeira aplicação de realidade aumentada teria sido feita em 1981, o *Super Cockpit*, utilizado em simuladores da força

aérea americana, passou a ter outro propósito como capacete para aumentar a visão do avião (como indicação de misseis disponíveis) em um projeto de milhões de dólares.

Por fim destacam-se como fatos importantes:

- O *Workshop on Augmented Reality and Ubiquitous Computing*, em 1993 no MIT, com a presença de pesquisadores de renome que ajudaram a popularizar a área.

- A disponibilização do *software* livre ARToolKit [16] baseado em rastreamento por vídeo, para desenvolvimento de aplicações em RA, o qual despertou interesse mundial.

- O I Workshop de Realidade Aumentada (WRA 2004), em Piracicaba, SP, precursor de workshops de RA e RV em todo Brasil [18].

Atualmente, o estado da arte da realidade aumentada se encontra em aplicações que envolvem sistemas inteligentes que reagem não deterministicamente (de acordo com a situação), sistemas que se utilizam de computação ubíqua e dispositivos vestíveis. Assim, com a evolução da internet e a redução de custo (e aumento do desempenho) dos computadores, aplicações de RA se tornam mais populares, impulsionadas pelo fácil acesso de customização (programação) e de sua execução em plataformas de uso comum como dispositivos móveis [17].

Conclui-se esse resumo histórico da RA, com o diagrama de Milgram [20] (Figura 2), que afirma que a realidade aumentada faz parte da chamada realidade misturada, e se posiciona em algum ponto da realidade-virtualidade contínua, que conecta um ambiente totalmente real a um ambiente totalmente virtual.

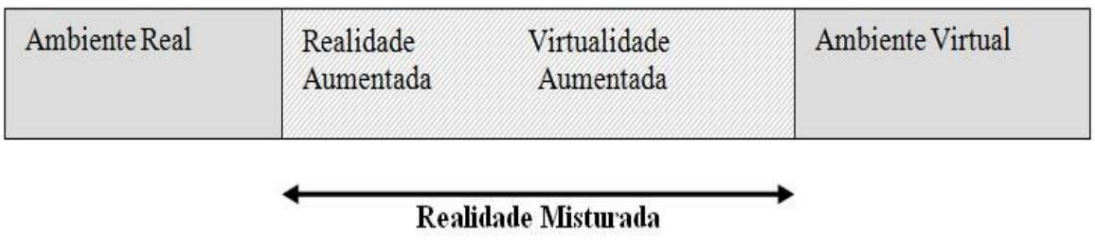

Figura 2: Diagrama de Milgram (Adaptado de [20])

2.2. Ferramentas de Suporte a RA para Dispositivos Móveis

A seguir são listadas algumas das ferramentas que auxiliam o desenvolvimento de aplicações de realidade aumentada, das quais, o Vuforia e o libGDX foram utilizados diretamente neste trabalho.

2.2.1. OpenGL

A *Open Graphics Library* (OpenGL) [43] é uma interface de programação de aplicações (API) multi-plataforma e multi-linguagem para renderização de gráficos vetoriais 2D e 3D. A API é tipicamente usada para interagir com uma unidade de processamento gráfico (GPU), para obter processamento acelerado por hardware.

As ferramentas que serão apresentadas ARToolkit, Vuforia e libGDX utilizam OpenGL como base para seu processamento abstraindo o usuário da programação de nível próximo ao *hardware*.

2.2.2. ARToolkit

O ARToolkit, de *Augmented Reality Toolkit*, é uma biblioteca de código aberto (licença LGPL v3.0) de rastreamento por vídeo, capaz de calcular a posição e orientação real da câmera relativa a marcadores quadrados físicos (seção 2.3) ou a marcadores de aspecto natural (imagens planas) em tempo real.

O ARToolkit foi desenvolvido originalmente por Hirokazu Kato do *Nara Institute of Science and Technology* em 1999 [16] e foi lançado ao público pela Universidade de Washington. Ele foi uma das primeiras bibliotecas a ser disponibilizada para dispositivos móveis, rodando no Symbian em 2005, e em 13 de maio de 2015 foi re-lançada após sua aquisição pela empresa de RA DAQRI com partes que antes só se encontravam na versão profissional paga, agora em código aberto.

A biblioteca é otimizada para dispositivos móveis e conta com suporte a OpenGl ES2.x, integração com GPS e bússola e calibração automática de câmera. Códigos de

exemplo, assim como suporte aos sistemas operacionais e detalhes específicos podem ser encontrados em sua documentação [5].

2.2.3. Vuforia

O Vuforia, assim como o ARToolkit, utiliza-se de visão computacional para reconhecimento (e ratreamento) de imagens planares e reconhecimento de múltiplos marcadores, porém diferentemente de seu concorrente, ele também reconhece formas 3D simples como cubos e cilindros em tempo real, além de incluir em suas funcionalidades a detecção de oclusão, que reconhece se um marcador está parcialmente omitido e a habilidade de criar e reconFigurar marcadores programaticamente durante a execução do programa [24].

Entretanto, apesar de seus grandes avanços, o Vuforia é de licença privada sendo que para usá-lo é necessário adquirir o produto ou, para usá-lo gratuitamente, mostrar uma marca d'agua durante a execução do aplicativo.

A biblioteca, desenvolvida originalmente pela Qualcomm, foi comprada em novembro de 2015 pela PTC Inc. e é considerada a mais avançada na indústria atualmente, [22] possuindo foco em dispositivos móveis e vestíveis (como óculos digitais), e suportando um número surpreendente deles. Porém, ao contrário do ARToolkit, não suporta sistemas operacionais de computadores pessoais como Windows e Linux.

Assim, a Vuforia provê suporte a linguagens C++, Java, Objective-C e .Net por meio de sua extensão a biblioteca de desenvolvimento de jogos Unity, além de prover suporte nativo ao Android e iOS. Para o suporte nativo ao Android é necessário baixar o arquivo *jar* da biblioteca e vincula-lo com o projeto. Como o Vuforia utiliza código nativo do Android e OpenGl ES é necessário que se vincule a NDK (*Native Development Kit*) ao projeto também, diferentemente de aplicações padrão que só utilizam a SDK (*Software Development Kit*) do Android em Java. A arquitetura mais detalhada da integração da biblioteca com o projeto Android se encontra na seção 3.2.3.

Por estas razões, apesar de o ARToolkit ser de código aberto, foi escolhido o Vuforia para a implementação do projeto.

2.2.4. LibGDX

A libGDX é uma biblioteca *open-source* de desenvolvimento de games escrita em Java, com alguns componentes dependentes de desempenho escritos em C/C++. Ela permite o desenvolvimento de aplicações multi plataforma (Windows, Linux, Mac OS X, Android, iOS, BlackBerry e navegadores com suporte a WebGL) utilizando o mesmo código de base. Ela foi criada por Mario Zechner a partir de um projeto pessoal (de 2009) disponibilizado na internet em licença aberta e se tornou popular pela sua simplicidade após a inclusão do suporte a física em dispositivos móveis [21].

A meta multi plataforma da biblioteca é atingida por meio de sua arquitetura que abstrai as diferenças entre aplicações de diferentes plataformas enquanto mantém o mesmo código base em Java para realizar as mesmas tarefas, como renderizar uma forma geométrica. Assim, códigos dependentes de plataforma (chamados de *backends*) ficam em pastas diferentes, porém códigos referentes a aplicação se encontram na mesma pasta. Um exemplo, é a declaração da atividade principal do Android (ver seção 2.4.2) que fica em uma pasta especifica do Android, esta atividade então chamará o construtor do programa encontrada na pasta comum (multi plataforma) ao inicializar.

Desta maneira, para se utilizar de bibliotecas específicas de uma determinada plataforma, recomenda-se que se crie uma interface para este módulo utilizando o padrão de projeto “fachada” (seção 2.6) e implemente-se o código de plataforma especifica para cada alvo (Android, Windows, etc.).

Para se ter uma visão geral de como a plataforma funciona, ressalta-se dois pontos: ciclo de vida da aplicação e os módulos disponibilizados [19].

Ciclo de Vida da Aplicação

Assim como no Android (seção 2.4.2) o libGDX provê métodos para se gerenciar os estados da aplicação (Figura 5) por meio de uma interface chamada de *ApplicationListener*. Estes métodos, então, são chamados quando algum evento ocorre. Uma breve explicação deles se encontra a seguir:

- *create()*: método chamado quando a aplicação é iniciada

- *resize(int width, int height)*: método chamado toda vez que a tela é redimensionada e a aplicação não está pausada. Ele também é chamado uma vez após o *create()*.

- *render()*: método chamado pelo laço principal do "jogo" por meio da aplicação toda vez que a renderização for ocorrer.

- *pause()*: no Android esse método é chamado sempre que o botão "Home" é pressionado ou a aplicação perde o foco. No *desktop* ele é chamado apenas antes do método *dispose()* antes da aplicação ser terminada.

- *resume()*: este método é chamado apenas no Android quando a aplicação retorna do estado de pause.

- *dispose()*: método chamado quando antes da aplicação ser terminada.

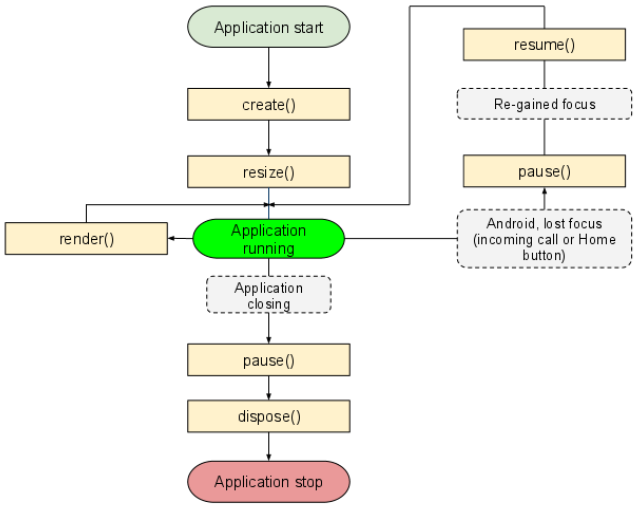

Figura 3: Ciclo de vida de um programa estruturado pelo libGDX (Retirado de [19])

Módulos

A libGDX possui diversos módulos que ajudam em cada etapa na construção de uma arquitetura de *game* simples.

- *Input*: provê um modelo unificado de entrada para todas as plataformas, suportando ações de teclado, mouse, toque e acelerômetro. As entradas são tratadas por meio de eventos registrados;

- *Graphics*: Abstrai a comunicação com GPU (unidade de processamento gráfica) e inclui uma implementação do OpenGl ES com métodos para se obter funcionalidades do mesmo;

- *Files*: abstrai o acesso a arquivos em todas as plataformas providenciando métodos convenientes para leitura e escrita independente da mídia;

- *Audio*: facilita a gravação e reprodução de áudio em todas as plataformas;

- *Networking*: providencia métodos para executar operações de redes, como requisições simples ou comunicação entre cliente e servidor;

A Figura 4 mostra os módulos em uma arquitetura de *game* simples.

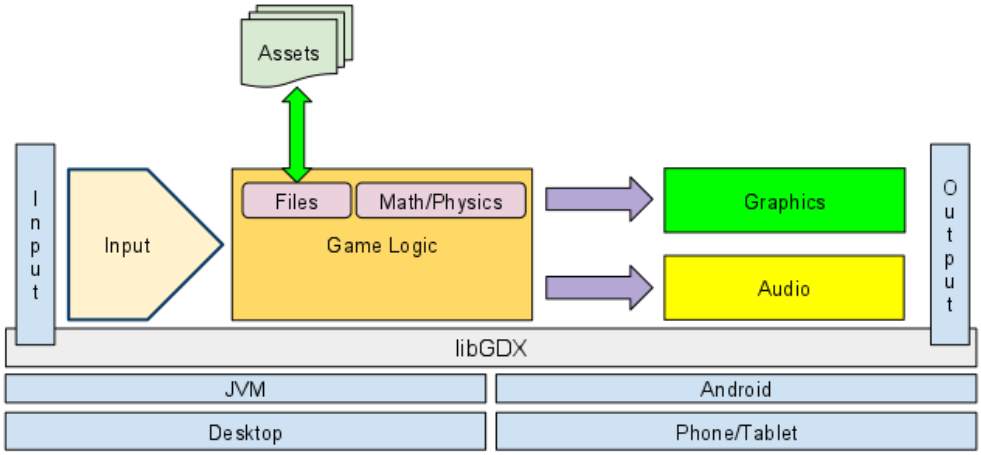

Figura 4: Funcionamento simplificado da libGDX (Retirado de [19])

2.2. Detecção de Marcadores

Uma das modalidades de realidade aumentada é o uso de marcadores como referência aos objetos que serão sobrepostos ao cenário real. Marcadores são identificadores retangulares geralmente constituídos por uma margem branca, borda preta e de um identificador qualquer em seu interior, podendo ser um símbolo, letras, ou uma imagem simples como um código de barras, sendo de fácil identificação em termos computacionais (Figura 6). O uso de marcadores envolve diversos passos que vão desde a detecção do marcador até a renderização do objeto. Esses passos formam a cadeia de RA (Figura 5) [25] [13].

Figura 5: Cadeia de realidade aumentada para detecção de marcadores. Fonte: [25]

Tem-se como exemplo alguns passos para detecção de marcadores do ARToolkit: antes de entrar na cadeia o ARToolkit exige que a câmera seja calibrada para a aquisição de parâmetros intrínsecos assim como fatores de distorção que formam a matriz de projeção (equação 1), onde *fx* e *fy* são as distancias focais, *s* é o tamanho efetivo do pixel, *u* e *v* são a deslocamento do ponto principal. A matriz de projeção é utilizada para translação de coordenadas do mundo para coordenadas da imagem. O marcador preto e branco deve estar salvo para uso posterior.

$$A = \begin{bmatrix} sf_x & 0 & u \\ 0 & sf_y & v \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

A primeira fase da cadeia, denominada “procurar marcadores” é a binarização da imagem, que é a transformação da imagem colorida em apenas dois níveis de intensidade: preto e branco. A vantagem da binarização é que se diminui a quantidade de memória utilizada para armazenar a imagem e também que se destaca apenas o que é de interesse ao algoritmo. Nesta fase aplica-se um limiar no valor de intensidade de cada *pixel*, assim, *pixels* com valores acima do limiar serão considerados brancos, e os abaixo, pretos.

A próxima etapa ainda dentro da fase “procurar marcadores” é a detecção de vértices e arestas para o reconhecimento de retângulos, nesta etapa identifica-se quando a quantidade de *pixels* varia bruscamente. É recomendável que se utilize o algoritmo *Zero Crossing* de segunda derivada com uma máscara de convolução discreta [12]. Após isto

detecta-se quais formas formam um contorno fechado e é realizada uma análise dos ângulos entre os vértices dos contornos para a classificação poligonal da forma [35].

O próximo passo, denominado “encontrar posição e orientação do marcador 3D”, existe devido a esses retângulos não necessariamente pertencerem ao plano da imagem e, então, é preciso transformá-los para que o marcador esteja no plano da tela, logo, aplica-se correção da distorção perspectiva para ajusta-los [26]. Assim, a equação 1 pode ser utilizada para transladar e escalar o marcador identificado para o plano da imagem.

O terceiro passo é denominado “identificar marcadores”. No caso do identificador do *ARToolkitPlus* (Figura 6), é feita uma varredura vertical e horizontal para retirar informações da malha 6x6 representada por 36 *bits* e utiliza-se a técnica de *subsampling* para diminuir os erros associados à homografia causada por erros de cálculo. No *subsampling* captura-se o valor de 9 pontos para cada *bit* sendo que o valor final é obtido por meio da média.

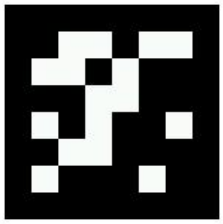

Figura 6: Exemplo de marcador da ferramenta de RA ARToolkitPlus

No quarto passo, denominado “posicionar e orientar objetos”, utiliza-se pontos de vértice nas coordenadas do marcador e pontos nas coordenadas da imagem para descobrir, de maneira interativa, os componentes de rotação e translação mais próximos aos parâmetros extrínsecos da câmera (equação 2), onde R_{ij} representa os parâmetros de rotação, T_i representa os parâmetros de translação e X_M, Y_M, Z_M representam a posição no mundo.

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_M \\ Y_M \\ Z_M \\ 1 \end{bmatrix} \quad (2)$$

Por fim, no passo denominado “renderizar objetos 3D no *frame* de video”, o OpenGL utiliza a matriz de transformação estimada (equação 2) para sobrepor os objetos virtuais ao quadro do vídeo.

2.4. Características do Android

O Android é atualmente o sistema operacional (SO) mais utilizado em dispositivos móveis, ocupando em média 83,7% dos aparelhos no último ano, seguido do iOs com 14,8% segundo o IDC (*International Data Corporation*) [14]. Foca-se nesta seção as peculiaridades do desenvolvimento para *Android* em relação a um sistema operacional de computadores pessoais.

2.4.1. Arquitetura

A começar pela arquitetura (Figura 7) [3], percebe-se algumas diferenças em referência aos SOs comums. Fundamentando-se no *Kernel* (núcleo) do Linux, o Android se aproveita de características de segurança chaves e também permite que fabricantes de dispositivos desenvolvam *drivers* para um *kernel* bem conhecido.

Já o HAL, camada de abstração de hardware, provê interfaces padrão aos dispositivos de hardware (como câmera ou *bluetooth*) à API (*Application Program Interface*) Java de mais alto nível. Por sua vez o Android *Runtime* (ART) provê a capacidade de executar múltiplas máquinas virtuais em dispositivos com pouca memória, sendo que a partir da versão 5.0 Android cada aplicativo possui seu próprio processo e sua própria instância da máquina virtual. Além disso, ele possui algumas características como compilação *just-in-time* (JIT) e *ahead-of-time* (AOT), *garbage collection* otimizado e inclui várias bibliotecas Java 8 bem estabelecidas.

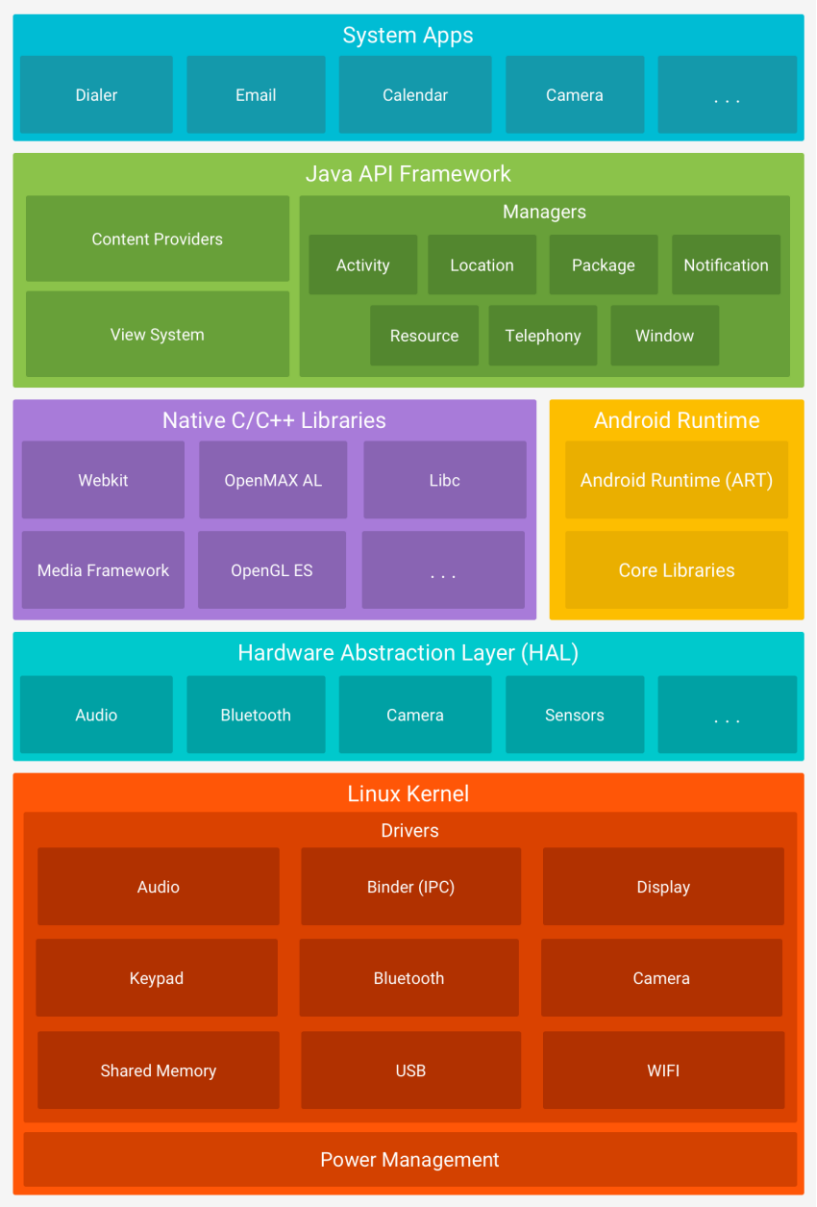

Figura 7: Arquitetura do sistema operacional Android (Retirado de [3])

As bibliotecas em C/C++ nativas do Android são utilizadas pelos componentes de sistema e serviços do Android, sendo que o Java API Framework expõe muitas dessas bibliotecas por meio de uma interface Java, porém se o aplicativo precisar utilizar código em C/C++ nativo ele pode se utilizar do NDK do Android.

Por fim, o Java API Framework disponibiliza todas as APIs necessárias para se desenvolver os aplicativos de Android, o que inclui: um sistema de interface de usuário rico e extensível, um gerenciador de recursos, um gerenciador de notificações, um

gerenciador de ciclo de vida e um provedor de conteúdos que permite que o aplicativo acesse dados de outros aplicativos, como o aplicativo de contatos, por exemplo.

2.4.2. Componentes dos Aplicativos

É importante salientar que um aplicativo Android difere-se de um programa comum por uma série de peculiaridades que são embutidas até mesmo aos aplicativos mais simples.

A começar, o SO atribui um identificador (ID) de usuário a cada aplicativo, então, o sistema define permissões de acesso para todos os arquivos do mesmo, de forma que somente o ID de usuário do aplicativo pode acessá-los. Por consequência, o Android implementa o princípio de privilégio mínimo pois o aplicativo tem acesso apenas aos componentes necessários para funcionar e não pode acessar componentes do sistema sem permissão.

A seguir, salienta-se quatro tipos de componentes fundamentais na construção de aplicativos para Android [4], sendo que cada um desempenha uma função específica, são eles:

Atividades

As atividades representam uma tela única do aplicativo, sendo que cada "atividade" é independente entre si, porém funcionam juntas para formar uma experiência coesa. É importante entender como uma atividade funciona, pois são o ponto de entrada para o programa e são elas que fornecem a interface com o usuário, assim, na Figura 8 é ilustrado o ciclo de vida de uma atividade.

Basicamente a atividade possui quatro estados:

- Execução: quando o aplicativo está executando em primeiro plano;
- Pausada: se a atividade perdeu o foco, mas está parcialmente visível. Uma atividade em pausa está completamente viva, porém o sistema pode matá-la em caso de pouca memória;

- Parada: se a atividade está completamente obscurecida por outra. Informações como o estado da atividade ainda continuam na memória, porém o aplicativo irá provavelmente ser retirado de memória caso outro aplicativo necessite de memória;

Quando a atividade está pausada ou parada, o sistema pode retirar a atividade da memória pedindo a ela para terminar ou parando seu processo. Assim, quando a atividade retorna ao primeiro plano é preciso inicia-la completamente novamente e restaurá-la para o estado anterior.

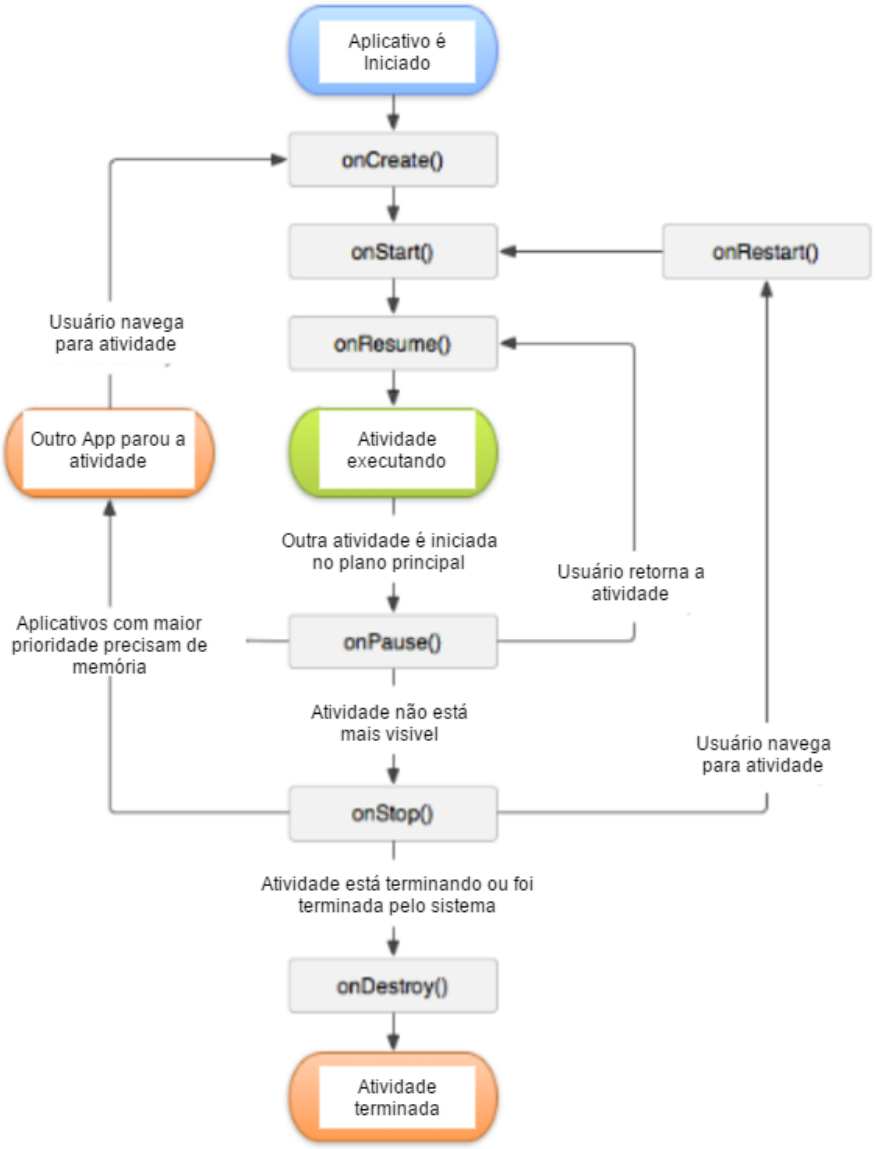

Figura 8: Ciclo de vida de uma atividade (Adaptado de [4])

Desta forma, o Android fornece métodos para tratar o ciclo de vida da aplicação em qualquer estado, como por exemplo, salvar dados do usuário quando a aplicação é fechada.

Serviços

Os serviços são componentes que atuam em plano de fundo para exercer operações longas ou para trabalhar com processos remotos, sendo assim, eles não apresentam uma interface do usuário. Um exemplo de serviço seria tocar músicas em segundo plano enquanto o usuário está utilizando algum aplicativo diferente.

Provedores de Conteúdo

Provedores de conteúdo são gerenciadores de um conjunto de dados do aplicativo, como o sistema de arquivos, um banco de dados ou qualquer local de armazenamento persistente que o aplicativo consegue acessar. Um exemplo é o provedor de conteúdo que gerencia as informações de contato do usuário, assim, qualquer aplicativo com as permissões adequadas pode consulta-los.

Ademais, os provedores de conteúdo são utilizados também para ler e gravar dados privados no aplicativo.

Receptores de transmissão

Os receptores de transmissão são componentes que respondem a eventos do sistema ou de outro aplicativo. Transmissões do sistema representam a maioria das transmissões e podem ser uma notificação de que a tela foi desligada ou a bateria está baixa, por exemplo. Já as transmissões dos aplicativos podem comunicar a outros aplicativos que um *download* terminou e dados estão disponíveis para uso, por exemplo.

Receptores de transmissão não possuem nenhuma interface com usuário, porém, eles podem criar uma notificação na barra de *status* para alertar o usuário de uma transmissão.

É interessante notar que as atividades, serviços e receptores de intenção são ativados por meio de uma mensagem assíncrona, chamada mensagem de intenção, que pode conter dados a serem transmitidos para o componente ou não. Já o provedor de

conteúdo é ativado por meio de uma notificação de um *ContentResolver* (resolvedor de conteúdo), o qual manipula todas as transações com o provedor.

Tendo em vista a organização dos componentes, é obrigatório que cada aplicativo possua um Arquivo de Manifesto, este arquivo é sempre lido antes da inicialização do aplicativo e contém a declaração de todos os componentes que serão utilizados pelo mesmo. Além disso, o arquivo de manifesto também identifica as permissões de usuário que o aplicativo precisa, como acesso à internet; ele declara recursos de hardware e software exigidos para o funcionamento do aplicativo, como câmera ou tela multi-toque; e as APIs que são utilizadas pelo aplicativo (além das APIs de *Framework* do Android), como a biblioteca do Google *Maps*.

2.5. Metodologia Ágil

O desenvolvimento ágil é uma maneira diferente de gerenciar times e projetos de TI (tecnologia da informação). Ele surgiu do "manifesto ágil", em 2001, após um grupo de funcionários discutirem sobre o método tradicional em vigor naquela época que estava falhando frequentemente, assim, eles criaram o manifesto ágil, que descreve quatro valores relevantes ao desenvolvimento de *software*, são eles:

- Indivíduos e interações acima de processos e ferramentas;
- *Software* funcional acima de documentação extensiva;
- Colaboração do cliente acima de negociações de contrato;
- Responder a mudanças acima de seguir planos.

Desta forma, desde 2001, o uso de métodos que suportam esses valores cresceu rapidamente e se tornou imensamente popular.

Kelly Waters descreve em seu livro *All About Agile* [33], sobre dez princípios chave da metodologia ágil, os quais ele acredita que são fundamentalmente diferentes da abordagem tradicional da metodologia *waterfall* [40], são eles:

- A participação ativa do usuário é imperativa;

- A equipe deve ter poder para tomar decisões;
- Requisitos evoluem, porém, o calendário é fixo;
- Capture requisitos de alto nível, visuais e em interações leves;
- Desenvolva pequenos módulos e incremente;
- Foque na entrega frequente de produtos;
- Complete cada modulo antes de começar o próximo;
- Aplique a regra 80/20 (foque nos 20% que irá dar mais resultados);
- Teste é integrado ao ciclo de vida do projeto, teste cedo e frequentemente;
- Uma abordagem colaborativa das partes interessadas é essencial.

Há várias metodologias que são conhecidas por serem ágeis como a DSDM (*Dynamic Systems Development Method*) [44], a mais antiga que surgiu antes mesmo do termo "ágil" ser inventado; o Scrum [45], que foca em como gerenciar as tarefas dentro de um time de desenvolvimento; e o XP [46] (*Extreme Programming*), utilizado neste projeto, o qual foca no processo de engenharia de software, abordagem do produto e fases de teste em ciclos pequenos de desenvolvimento.

2.6. Padrões de Projeto

Em engenharia de *software*, um padrão de projeto é uma solução geral e reutilizável para um problema comum e recorrente dentro do contexto de projetos. É interessante salientar que padrão de projeto não é um projeto final que pode ser transformado diretamente para código fonte ou de máquina, ele é, na verdade, uma descrição ou modelo de como resolver um problema. Nos padrões de projeto são formalizadas as melhores práticas que um desenvolvedor pode usar para resolver problemas comuns ao projetar um aplicativo ou sistema.

O termo "padrões de projeto" originou-se de um conceito arquitetural criado por Christopher Alexander em seus livros *A Pattern Language* (1977) [1] e *The Timeless Way*

of Building (1979) [2] em que ele define que um padrão deve possuir idealmente conceitos de encapsulamento, generalidade e abstração e devem ser extensíveis e flexíveis para serem combinados. Entretanto, apesar do termo ser definido em meados de 1979, ele se popularizou com o livro *Design Patterns: Elements of Reusable Object-Oriented Software* da "gangue dos quatro" com 23 padrões estabelecidos, organizados em três famílias: padrões de criação, padrões estruturais e padrões comportamentais [8] [23].

A seguir destaca-se alguns dos padrões que foram utilizados neste projeto.

Singleton

Padrão de projeto de criação que garante a existência de apenas uma instância da classe, mantendo um ponto global de acesso ao seu objeto. Seu diagrama UML (Unified Modeling Language) é apresentado na Figura 9.

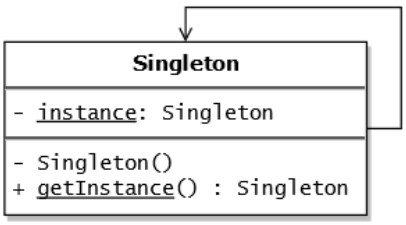

Figura 9: Padrão de projeto Singleton (Adaptado de [47])

Prototype

Padrão de projeto de criação que permite a criação de novos objetos a partir de um protótipo que é clonado. Seu diagrama UML se encontra na Figura 10.

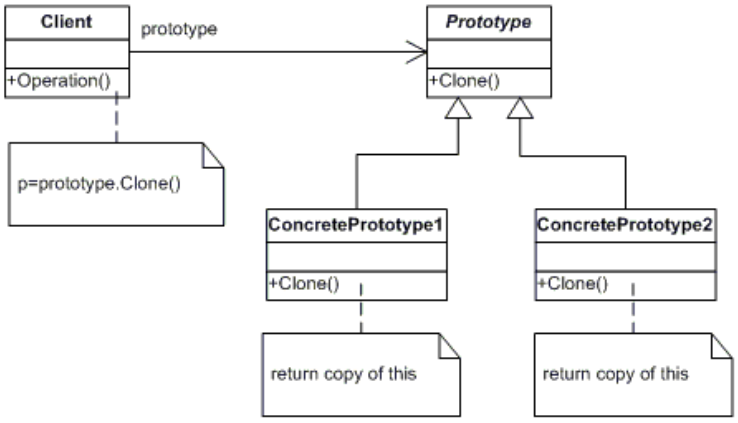

Figura 10: Padrão de projeto Prototype (Adaptado de [47])

Facade

Padrão de projeto estrutural que esconde as complexidades de um sistema maior e provê uma interface simplificada ao cliente. Um exemplo em UML encontra-se na Figura 11.

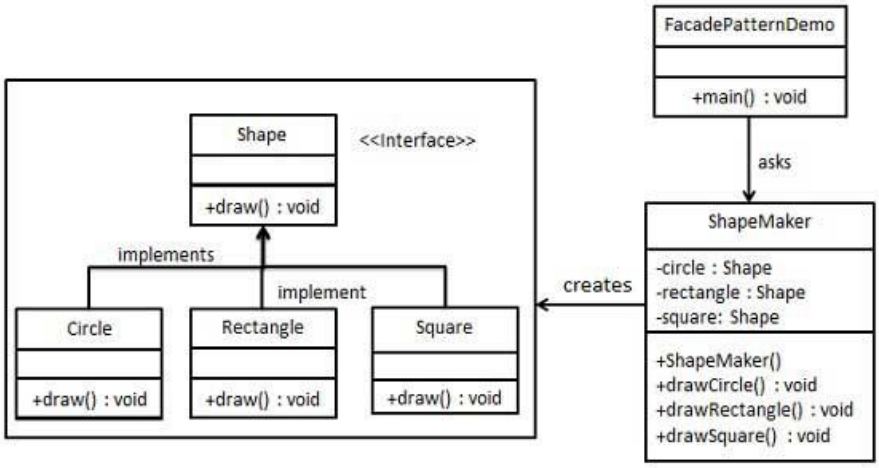

Figura 11: Padrão de projeto Facade (Adaptado de [47])

Flyweight

Padrão de projeto estrutural apropriado para situações em que vários objetos manipulados em memória possuem informações repetidas, assim, agrega-se a informação repetida em um objeto adicional que geralmente é implementado como um *singleton*. Uma ilustração utilizando *flyweight* em UML encontra-se na Figura 12.

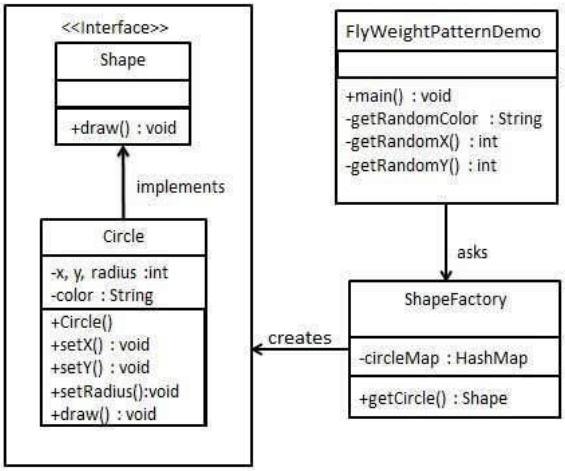

Figura 12: Padrão de projeto Flyweight (Adaptado de [47])

Command

Padrão de projeto comportamental em que um objeto encapsula a informação necessária para executar uma ação ou acionar um evento. Seu diagrama UML encontra-se na Figura 13.

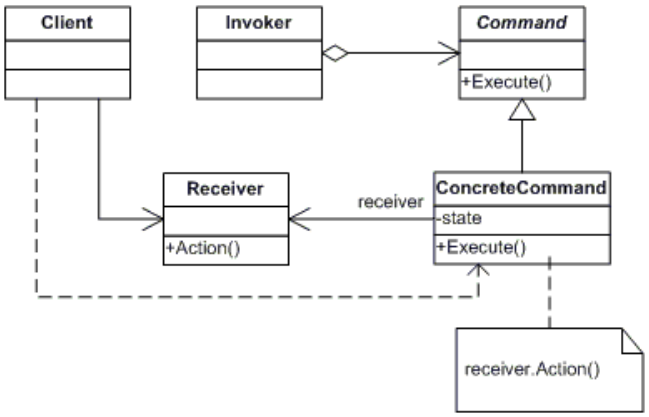

Figura 13: Padrão de projeto Command (Adaptado de [47])

Observer

Padrão de projeto comportamental que permite que objetos interessados sejam avisados de eventos ocorrendo em outro objeto. Seu diagrama UML encontra-se na Figura 14.

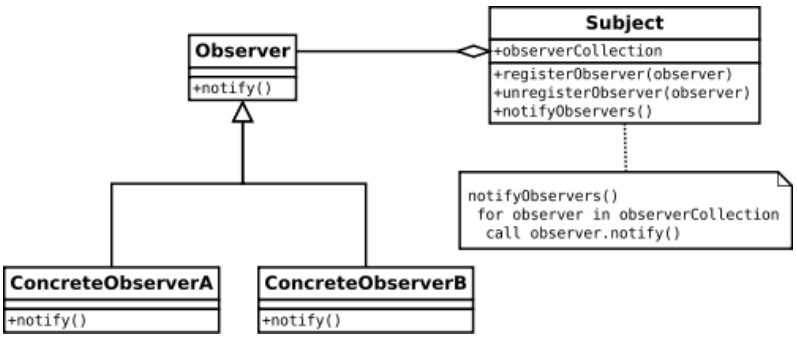

Figura 14: Padrão de projeto Observer (Adaptado de [47])

Template Method

Padrão de projeto comportamental que define um método concreto (*template*) que se utiliza de métodos abstratos em seu algoritmo, assim, as classes que estendem desta classe abstrata são obrigadas a implementar suas operações abstratas e definir um comportamento único para o método. Seu diagrama UML encontra-se na Figura 15.

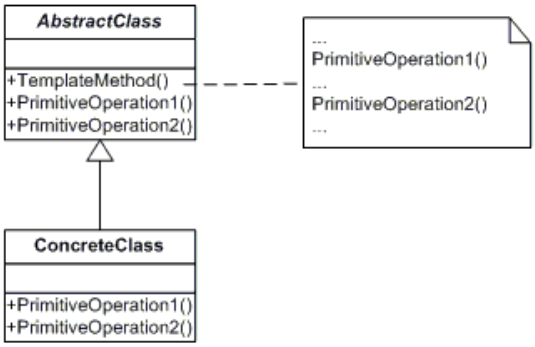

Figura 15: Padrão de projeto Template Method (Adaptado de [47])

2.7. Considerações Finais

Neste capítulo foram apresentados os conceitos utilizados no desenvolvimento da aplicação deste trabalho, desta forma, conclui-se que para se desenvolver este tipo de aplicação é necessário ter uma boa base em termos de projeto de *software* e computação gráfica, assim como é importante conhecer com certa profundidade as ferramentas utilizadas para melhor aproveita-las.

Portanto, espera-se que estes conceitos sejam de grande auxílio possibilitando uma leitura fluida ao próximo capítulo. Assim, no próximo capítulo, é apresentada a forma como esses conceitos foram aplicados e as dificuldades e facilidades encontradas durante a implementação deste projeto.

CAPÍTULO 3: DESENVOLVIMENTO DO TRABALHO

3.1. Considerações Iniciais

Na seção 3.2 são mostradas as etapas de planejamento e construção do trabalho e nas seções 3.4 e 3.5 são apresentados resultados e dificuldades encontradas no decorrer do desenvolvimento.

3.2. Projeto

Por meio da base desenvolvida sobre a revisão bibliográfica, construiu-se a aplicação, de forma que nas seções a seguir são encontradas as decisões de projeto, e a descrição do desenvolvimento das mesmas em ordem cronológica.

O diagrama da figura 16 ilustra o funcionamento do aplicativo de forma simplificada, indicando onde cada ferramenta atua.

3.2.1. Escolha das Ferramentas de Suporte a RA

Na época em que este trabalho começou a ser desenvolvido existiam algumas ferramentas que davam suporte a RA, porém, o ARToolkit e o Vuforia se destacavam em qualidade e tamanho da comunidade. Assim, ambos foram testados em relação a facilidade de uso e qualidade da projeção do objeto virtual sobre o marcador.

Em relação a facilidade de uso, ambos oferecem suporte nativo para Android, e possuem um modo de operação bem parecido, com ciclos de inicialização e renderização quase equivalentes. Entretanto, na época, o ARToolkit, ao contrário do Vuforia, não oferecia suporte a ferramenta mais recente de desenvolvimento de aplicações para Android (*Android Studio*), sendo mais complicada a sua integração com um projeto Android. Além disso, o Vuforia possuía, códigos e documentação mais claros sobre como projetar uma aplicação básica.

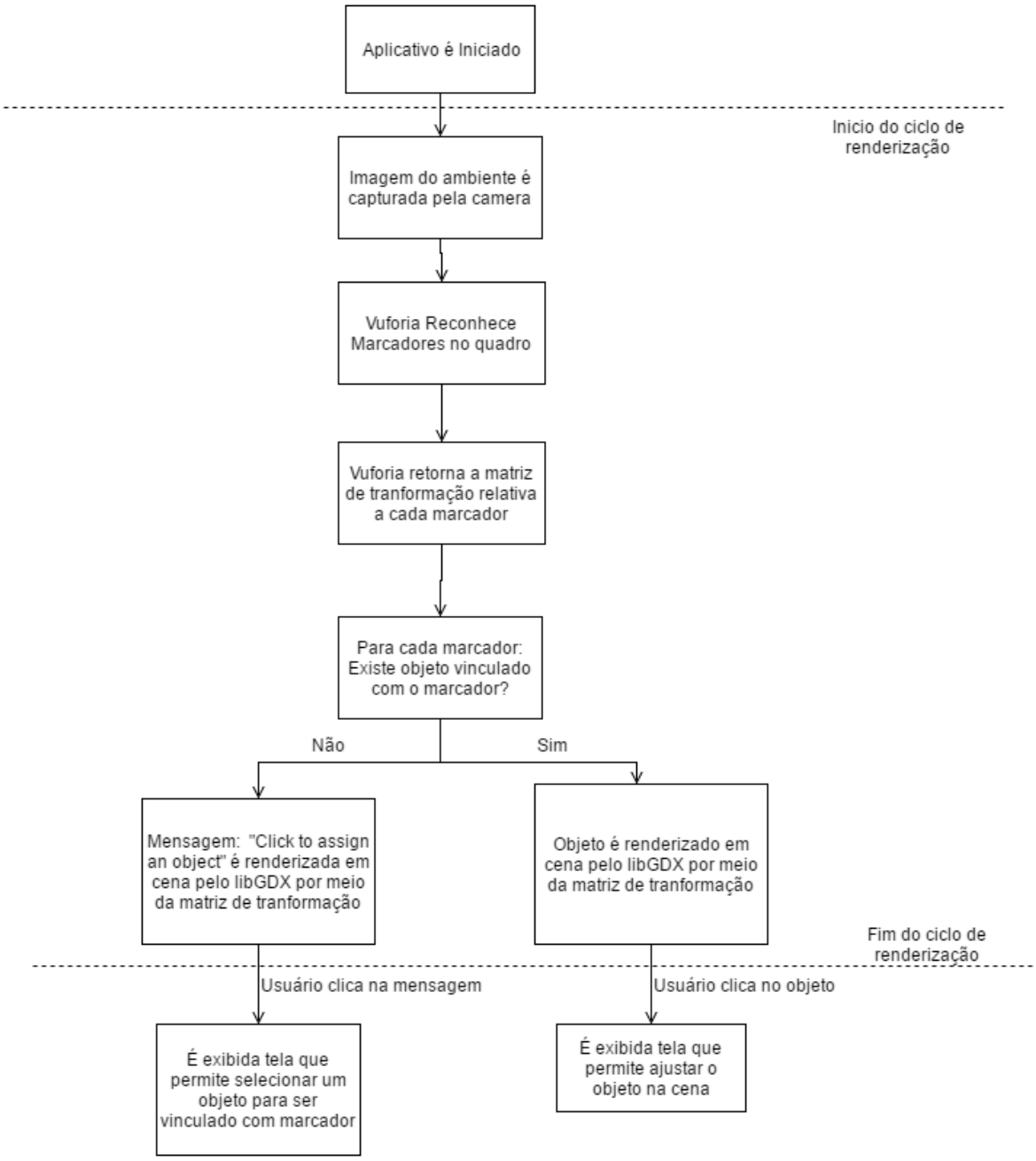

Figura 16: Fluxo básico da aplicação

Ademais, nos testes com marcadores, o Vuforia se mostrou mais preciso, sendo que o objeto se mostrou “fixo” sobre a superfície do marcador mesmo em condições mais adversas para a detecção do mesmo, como o marcador quase paralelo ao raio de visão da câmera ou a baixa luminosidade. Nos testes com o ARToolkit, o objeto se mostrava com pequenas vibrações nessas condições ou desaparecia (caso que acontece quando o marcador não é detectado).

Assim, apesar da grande vantagem do código aberto do ARToolkit, o Vuforia foi escolhido, pois o mesmo se alinha melhor com o propósito do projeto que é desenvolver um produto para o consumidor final além da facilidade de integração do mesmo com o Android Studio e a documentação mais acessível que motivaram esta escolha.

A segunda escolha a ser feita foi em relação a ferramenta de suporte gráfico. Como explanado na seção 2.2, o Vuforia possui suporte nativo ao Android, porém todas as outras atividades desde a estruturação do objeto até sua renderização na tela são feitas por meio do OpenGL ES, que apesar de auxiliar quem deseja trabalhar com ajustes gráficos específicos, atrasaria o projeto de uma aplicação completa, uma vez que deixar-se-ia de trabalhar em estruturas de projeto, para desenvolver-se vários algoritmos simples e que não possuem valor (como a leitura de um arquivo de objeto) para suportar uma aplicação básica.

A fim de evitar o trabalho maçante ao se utilizar o OpenGL ES, foi escolhida a ferramenta libGDX para o apoio gráfico, pela sua simplicidade, fácil integração com o Android Studio e por funcionar como uma biblioteca anexável ao código, ao contrário de alguns concorrentes do ramo que funcionam de forma totalmente gráfica e fugiriam do escopo do trabalho.

3.2.2. Planejamento

Para a realização deste trabalho escolheu-se o método de desenvolvimento ágil Extreme Programming (XP) por ser o mais adequado ao estilo do mesmo. Deste modo, alguns dos fatores que influenciaram esta escolha são baseados em práticas e princípios de XP:

- Projeto de apenas um desenvolvedor. Enquanto outros métodos como o *Scrum* são ótimos para equipes pequenas, o XP, pelo seu princípio de simplicidade, se encaixa bem quando há apenas um programador;

- Requisitos que mudam rapidamente. Por ser a primeira aplicação desenvolvida do tipo pelo aluno e do primeiro contato com algumas ferramentas espera-se que alguns requisitos mudem. Assim, segue-se a prática da refatoração que diz

que ela deve ser feita sempre que possível, buscando principalmente simplificar o código atual sem perder nenhuma funcionalidade;

- Entregas frequentes. Além de ser um objetivo do projeto, as entregas frequentes ajudam a validar pequenas partes do *software* por meio de protótipos e incrementá-los sempre que há necessidade. Além disso, outro objetivo da entrega frequente no XP é realizá-la com o menor tamanho possível, contendo os requisitos de maior valor para o negócio;

Assim, escolheu-se um conjunto básicos de requisitos, considerados de maior valor para dar início ao desenvolvimento: ao se apontar a câmera para um marcador o Vuforia deveria reconhecê-lo e obter a matriz de transformação, assim, um móvel pré-selecionado deveria aparecer sobre o mesmo por meio da renderização feita pela libGDX com ajuda da matriz obtida. Desta forma, o protótipo inicial assim como os objetivos fundamentais são descritos na próxima seção. E, no decorrer do texto, são descritas as mudanças e incrementos feitos a partir de avaliações do produto.

3.2.3. Integração das Ferramentas com o Android

Após a escolha das ferramentas, foi planejado e implementado um protótipo simples para verificar o funcionamento das duas ferramentas operando em conjunto, devido ao Vuforia não possuir nenhum *plugin* específico para libGDX. Desta forma, o protótipo deveria cumprir dois requisitos simples: ler um arquivo contendo a especificação do objeto e exibir o objeto em cima do marcador.

Assim, o objetivo do protótipo, além de cumprir os requisitos, seria integrar as duas ferramentas mantendo seu funcionamento independente e comunicando-se apenas quando necessário. O seu diagrama de sequência simplificado se encontra na Figura 17.

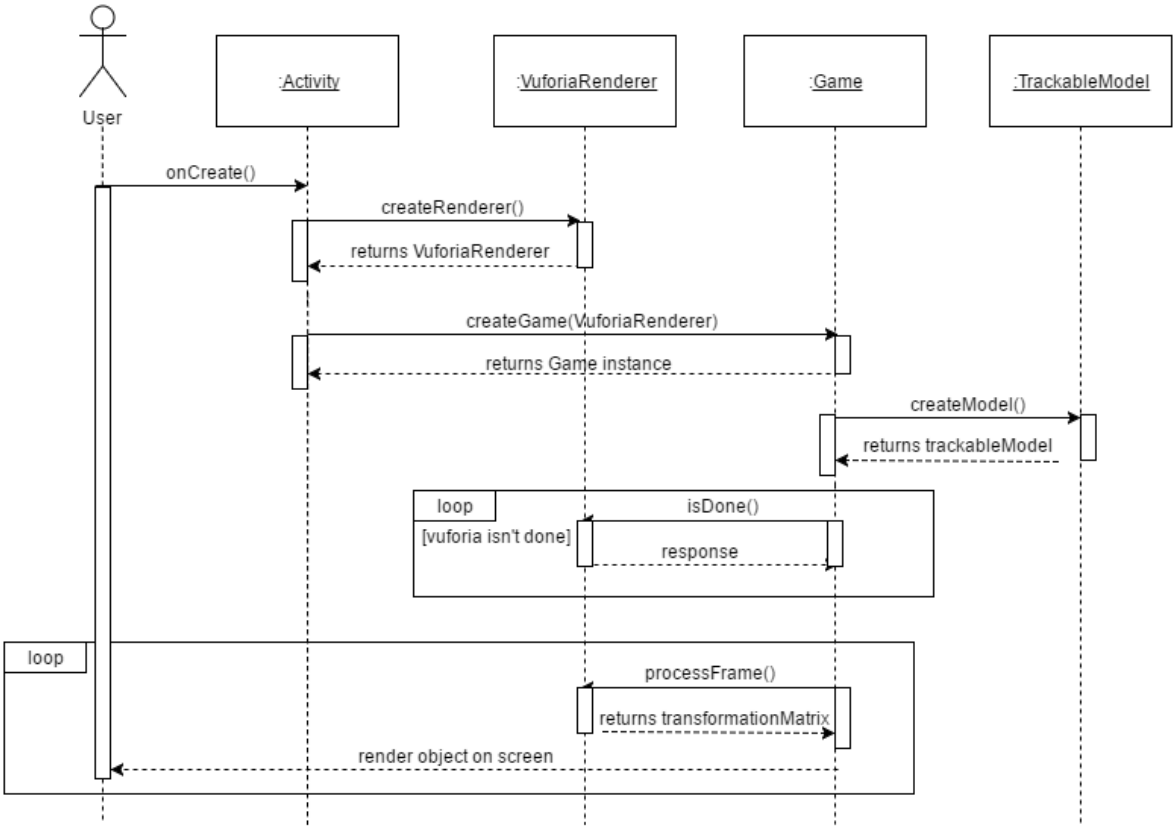

Figura 17: Diagrama de sequência da integração das ferramentas Vuforia e libGDX

Como explicado na seção 2.2 o libGDX separa de maneira lógica o código de *backend* de Android do código do programa geral. A classe principal do libGDX onde o programa é inicializado é chamada de *Game*, assim neste trabalho obedeceu-se a nomenclatura estipulada. Desta forma, a aplicação é iniciada por meio do método *onCreate* do Android, em seguida são inicializados os módulos do Vuforia responsáveis pela seção do mesmo e pelo processamento de imagem. Após isto, inicializa-se o *game* (ou "jogo", módulo raiz do libGDX) e espera-se o Vuforia acabar de inicializar antes de seguir com o ciclo de renderização, enquanto isso, lê-se o objeto a ser exibido de um arquivo.

Após a inicialização dos componentes entra-se no *loop* de renderização do jogo, o qual constitui-se do seguinte: o Vuforia processa o quadro capturado, retornando uma matriz 4x4 do tipo (R_x, R_y, R_z, T) contendo a transformação linear correspondente a posição do marcador em relação a posição da câmera. Como o Vuforia define o sistema de coordenadas diferente do libGDX é necessário fazer uma pequena mudança na matriz antes de encapsulá-la em um objeto próprio do libGDX. Assim, após a obtenção da matriz de transformação, o objeto é exibido na tela em cima do marcador.

3.2.4. Projeto Inicial

Após a verificação do funcionamento das ferramentas em conjunto, elaborou-se uma arquitetura inicial que serviria de base para o restante da aplicação. A arquitetura deveria ser simples, porém bem padronizada para facilitar a posterior expansão do programa. Para tanto, definiu-se uma relação de um marcador para um objeto, desta forma, ao detectar-se o marcador o objeto equivalente deveria ser exibido sobre ele. A Figura 18 apresenta o diagrama de classes do projeto inicial em relação ao gerenciamento de objetos.

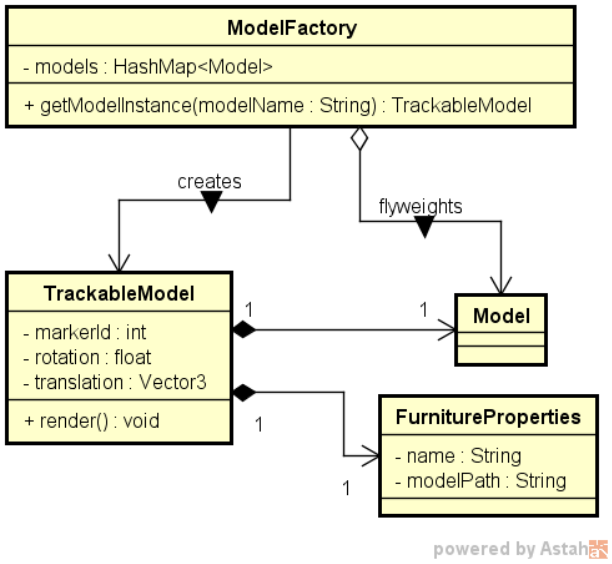

Figura 18: Projeto inicial do gerenciamento de objetos

Nota-se no diagrama que se utilizou do padrão de projetos *flywheight* mantendo apenas uma instância para cada tipo de modelo, assim, objetos iguais compartilham do mesmo modelo, o que gera economia memória e acarreta em um carregamento mais rápido a segunda vez que um objeto igual é criado, uma vez que seu modelo já estará em memória e então, só será feita a referência. Além disso, este padrão facilita o gerenciamento de memória pois a libGDX sendo implementada em C++ em seu núcleo deixa a cargo do programador a destruição dos modelos para a liberação de memória, assim, guardando todas as referências dos modelos na fábrica fica simples de gerenciá-los e de destruir todos ao final do programa.

Nota-se também que o objeto (chamado de *furniture* ou mobilia) é formado por três partes: a instância, que contém o identificador do marcador e medidas como translação e

rotação do objeto; o modelo, que contém os vértices e as texturas que serão renderizadas; e as propriedades, que contém o nome da mobília descrição e meta dados como o caminho do modelo. Para facilitar de adição e remoção de objetos na aplicação as propriedades são salvas em um arquivo JSON (JavaScript Object Notation) [42] que representa um banco de dados simples.

O banco de dados é ilustrado na Figura 19, assim, pode-se acrescentar de forma simples um objeto na aplicação. Por exemplo, para acrescentar uma cama, basta adicionar uma linha na tabela de mobílias com suas propriedades (categoria, nome, caminho do modelo, e outros) e adicionar o modelo da cama na pasta especificada, e a cama estará pronta para ser utilizada na aplicação. A implementação simplista com arquivos JSON se comporta da mesma maneira, bastando adicionar as propriedades como um objeto JSON na lista de mobílias.

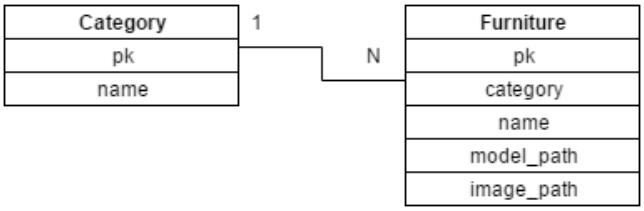

Figura 19: Representação do banco de dados da aplicação

Assim, o ciclo de renderização da aplicação foi implementado como no pseudo-código a seguir:

Tabela 1: Pseudo código do ciclo de renderização do aplicativo

```
lista_de_marcadores = obter_marcadores_visiveis()
Para cada objeto da lista_de_objetos:
    Se o marcador do objeto está em lista_de_marcadores:
        rederize(objeto, marcador)
        lista_de_marcadores.remova(marcador)
    Caso contrário:
        esconda(objeto)
Para cada marcador na lista_de_marcadores:
    novo_objeto = crie_novo_objeto(marcador)
    lista_de_objetos.adicione(novo_objeto)
```

3.2.5. Organização das Telas

Seguindo os preceitos da metodologia ágil pelo XP, as próximas seções de desenvolvimento foram baseadas em requisitos de alto nível de interface com usuário. Desta forma elaborou-se um sistema de telas para ajustar e selecionar uma mobília, o qual é apresentado na Figura a seguir.

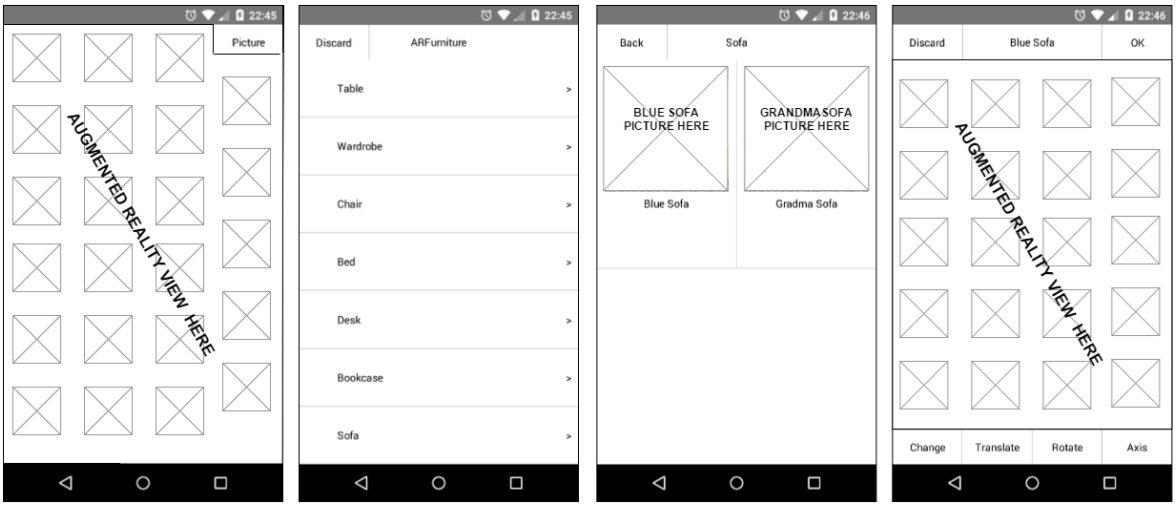

Figura 20: Organização das telas, da esquerda para direita: tela principal, seleção de categoria, seleção de mobília e tela de ajuste

Assim, foi elaborado o diagrama representando as ações que mudariam o estado da aplicação (Figura 21) e a partir desses requisitos foram implementadas as telas e funcionalidades restantes da aplicação. A começar pela tela de maior valor a de seleção de categoria e mobília, depois a de ajuste, depois a funcionalidade de botões mais importantes (como translação e rotação) e por fim funcionalidades menores como o botão de foto.

É interessante notar que no decorrer do projeto alguns requisitos mudaram para melhor se adequar a experiência do usuário, como o botão de descarte da tela de seleção de categoria que antes era um botão de voltar/salvar mas que após testes foi substituído pela ação de descarte por se encaixar melhor à situação, porém graças a modularização do código e aos padrões de projeto utilizados mudanças como esta são de fácil ajuste.

Na Figura 21 é apresentado o diagrama de fluxo de telas. A partir dele previu-se o uso de alguns métodos (e comportamentos) que foram implementados posteriormente. Na aplicação sempre que se deixava de operar com a realidade aumentada, nas ações (1) e (6),

era enviada a mensagem *pauseAR* para o Vuforia com intuito de evitar processamento desnecessário enquanto o usuário navega pelos menus. Da mesma forma, ao se retornar para as telas principal ou de ajuste, pelas ações (2) ou (5), era enviada a mensagem *resumeAR* para o Vuforia. Para se conservar o estado do objeto, um clone do mesmo era criado (seção 2.6) a partir das ações (9) e (5), desta forma se o usuário desistisse da mudança, por meio das ações (2) ou (8), o objeto original poderia ser restaurado, porém se o usuário aprovasse a mudança por meio da ação (7) o clone seria descartado e o objeto modificado conservado. Nota-se que as mudanças (3) e (4) não fazem nada além de chamar a tela especificada.

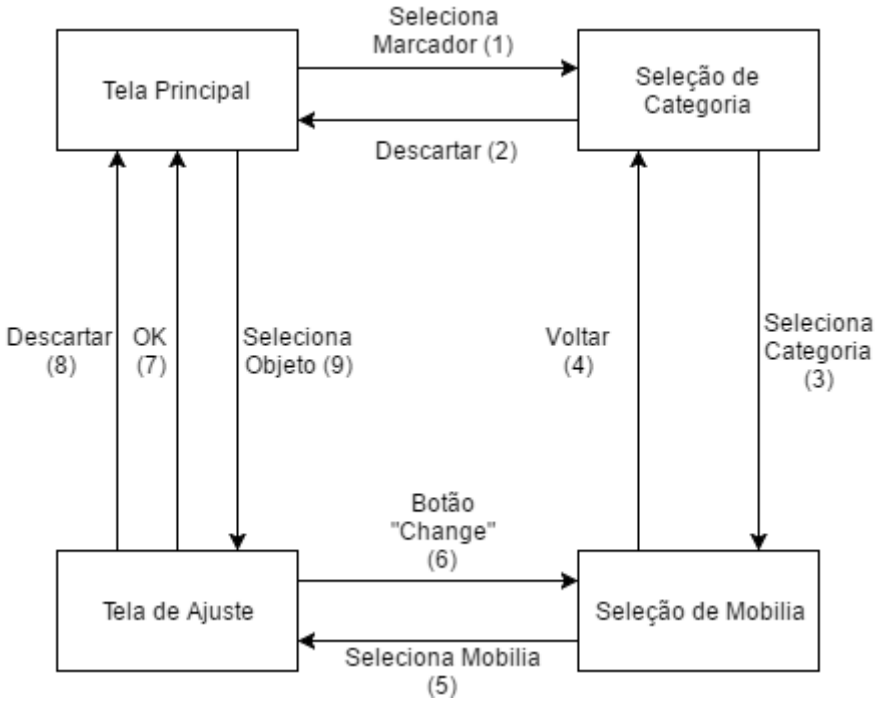

Figura 21: Diagrama do fluxo de telas

3.2.6. Seleção de Objetos por Toque

Para a melhor experiência do usuário, estipulou-se que a seleção de objetos seria feita por meio de toque. Apesar da libGDX possuir uma extensão que supre esta necessidade ela não foi utilizada para não aumentar o tamanho do arquivo do instalador, assim foi implementado um algoritmo com auxílio de estruturas que a própria ferramenta já oferecia. Ele é apresentado a seguir:

- Calcula-se o raio de seleção, que é um vetor gerado por meio da posição tocada na tela e da direção em que a câmera aponta;

- Para cada objeto na cena, calcula-se o ponto de interseção entre o objeto e o raio;

- O objeto cujo ponto de interseção é o mais próximo a seu centro é selecionado;

O raio de seleção e o ponto de interseção são calculados por meio da biblioteca, porém seus funcionamentos são explicados a seguir.

Para cada coordenada de duas dimensões na tela há um número infinito de coordenadas 3D. Por exemplo, se a câmera está posicionada em (x = 0, y = 0, z = 0) no centro da tela apontando para -Z, logo, um objeto em (0, 0, -10) será desenhado na mesma posição (da tela) que um objeto em (0, 0, -20). Assim, a coordenada no centro da tela representa os pontos (0, 0, -10) e (0, 0, -20) assim como os infinitos pontos que passam na semirreta formada por estes dois pontos a partir do plano próximo da câmera. Esta semirreta é chamada de raio de seleção e é representada pelo ponto de origem, localizado no ponto do plano mais próximo a câmera, e um vetor de direção. Matematicamente o raio de seleção pode ser representado pela equação 3:

$$f(t) = origin + t * direction \quad (3)$$

Assim, ao se tocar na tela são adquiridas suas coordenadas. Estas são normalizadas a valores proporcionais a sua posição na tela e multiplicadas pela matriz de projeção inversa da câmera, sendo que para a origem é utilizado z = 0. Do mesmo modo é feito o cálculo para o vetor de direção, porém com z =1 e após, é subtraído o resultado obtido pelo ponto de origem.

Já para o cálculo da interseção entre o objeto e o raio, deve-se antes, ao criar-se um objeto, aproximar suas dimensões (altura, largura e profundidade) à dimensão da forma que mais se adequa ao mesmo, logo, neste trabalho supôs-se que a forma mais próxima da maioria das mobílias seria um paralelepípedo. O cálculo dos limites do objeto é custoso e, portanto, só é feito na inicialização e guardado para uso posterior.

Desta forma o cálculo da intersecção é simplificado, neste caso o paralelepípedo pode ser visto como um conjunto de 3 pares de planos paralelos, assim, para cada plano é calculada a interseção com o raio (e descoberto o parâmetro *t* da equação 3) e para cada par de plano (x_1 e x_2, y_1 e y_2, z_1 e z_2) são salvos o t_{min} e o t_{max}, deste modo se o menor dos t_{max} for maior que o maior dos t_{min} significa que o raio atravessou a caixa [34]. A Figura 22 exemplifica este processo.

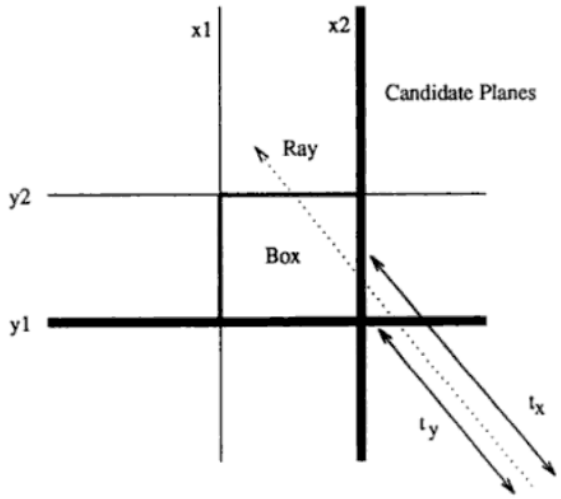

Figura 22: Interseção do raio com o paralelepípedo [34].

3.2.7. Arquitetura da Interface com Usuário

Para que sejam exibidas imagens na tela, processadas pelo *loop* principal da aplicação, o libGDX requer que sejam registradas telas (por meio da interface *Screen*) no objeto principal da aplicação chamado de *GameEngine*. Para trocar de tela, cria-se então uma nova tela, registra-a no jogo e destrói-se a tela anterior. A Figura 23 auxilia na compreensão da arquitetura de interface com o usuário por meio de um diagrama de classes.

Assim, tudo que é mantido no objeto da tela é perdido, sendo que instâncias importantes como a lista de objetos e o estado da aplicação são mantidos dentro de *GameEngine*. Para gerenciar estes objetos, o *game* possui instâncias estáticas das classes *ScreenManager* e *InstancesManager*, sendo que a primeira é responsável pela troca de telas e por manter informações como nome da tela anterior, e a segunda é responsável por gerenciar a lista de objetos e por manter um clone do objeto original ao se entrar na tela de

ajuste. O fato de esses objetos serem estáticos, deixa a implementação dos *listeners* dos botões mais simples, como será visto adiante.

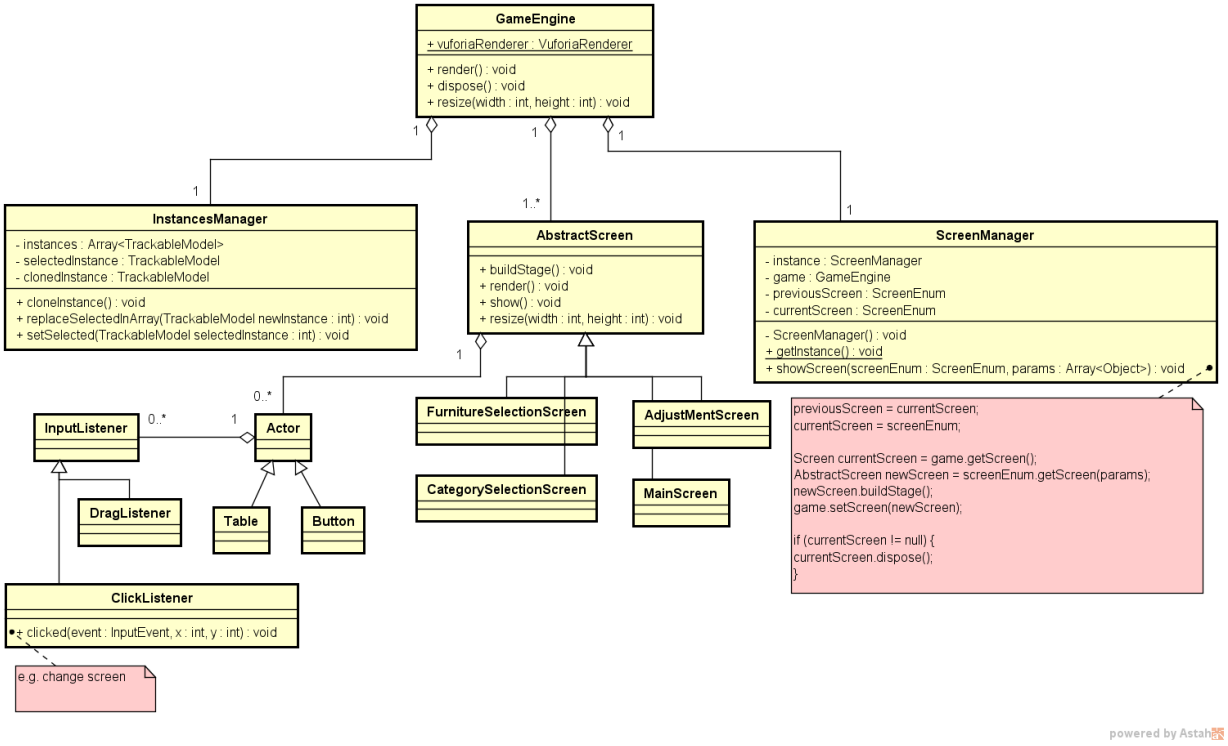

Figura 23: Diagrama de classe da arquitetura de interface com o usuário

Assim todas as telas da aplicação estendem uma tela abstrata chamada *AbstractScreen*, criada para simplificar a troca de telas contendo implementações de métodos utilizados por todas as classes filhas. A *AbstractScreen* implementada neste projeto tem como interface *Screen* (tela) e estende a classe *Stage* (palco). Por sua vez o palco aceita múltiplos *Actor* (atores) que são os elementos de interface com o usuário. No caso deste trabalho, os atores são o os botões, listas, rótulos dentro dos menus.

Os atores contém texturas de forma que foram organizados e inspirados pelo padrão *flywheight*, e como elementos de interface são sempre iguais, por exemplo, todos os botões da aplicação possuem fundo branco que fica azul quando pressionado, estas definições ficam guardadas todas em um único arquivo JSON e são carregadas por um único objeto da classe *Skin*, assim há facilidade para se gerenciar a aparência da interface e também para se trocar de estilos (pois basta carregar outro arquivo de *Skin* para isso).

Os atores, por sua vez, são capazes de receber e processar ações do usuário, por meio do registro de *listeners* que são objetos que contém métodos para serem executados quando determinado evento ocorrer. Desse modo, decidiu-se que todos os *listeners* seriam organizados na mesma classe, de forma a ser simples de se registrar e modificar cada *listener*.

Na tabela 2 é mostrado o exemplo de um *listener*. O primeiro é utilizado para se mudar para da tela principal para a tela de seleção de mobília, primeiro pausa-se o processamento de imagens do Vuforia e após nota-se a simplicidade para se mudar de tela por meio do método *showScreen*.

Tabela 2: Exemplo de *listener* utilizado para trocar a tela principal para a tela de seleção

```
public static ClickListener showMenuListener(final Category category) {
    return new ClickListener() {
        public void clicked(InputEvent event, float x, float y) {
            GameEngine.vuforia.pauseAR();
            ScreenManager.getInstance().showScreen(ScreenEnum.FURNITURE_MENU,
category);
        }
    };
}
```

3.3. Resultados Obtidos

A começar pela organização da interface com o usuário, a aplicação obteve um resultado positivo pelo menu de seleção se parecer com o de aplicações de venda de mercadorias as quais os usuários já estão acostumados a lidar. Nota-se que este foi planejado e retirado de especificações de interface da seção 3.2.5.

Além disso, o planejamento e a refatoração para melhoria de processos alavancou um ganho de desempenho de algumas operações matriciais no laço principal da aplicação. Também se destaca que o estudo e o cuidado ao se escolher as ferramentas e tecnologias utilizadas foram fundamentais para a agilidade do desenvolvimento e ajudaram a promover uma experiência natural em relação a realidade aumentada.

Por fim, o código modular e bem planejado facilitou a construção de novas telas e a adição de funcionalidades, sendo simples atualizar funcionalidades e refatorar pequenas porções de código.

Deste modo se comparado a aplicativos do ramo de decoração como o *iStaging*, a aplicação desenvolvida possui maior estabilidade do objeto renderizado sobre o ambiente real, além de não necessitar de ajustes manuais. Em relação ao IKEA *Catalogue*, o aplicativo desenvolvido é capaz de mostrar mais de um objeto em cena, necessitando apenas de mais marcadores impressos, enquanto o IKEA *Catalogue* necessita de catálogos para referência.

Na Figura 24 pode-se ver o aplicativo resultante em funcionamento. Seu funcionamento segue o planejamento de interface com o usuário e o fluxo de telas explicados no decorrer da seção 3.2. Desta forma, ao se iniciar o aplicativo, se existir algum marcador enquadrado é exibida a mensagem em cima dele “Click to assign an object” como ilustra o primeiro quadro da figura.

Por conseguinte, ao usuário selecionar o marcador, é exibido o menu de seleção de categorias. Ao se selecionar uma categoria é exibido o menu de seleção de mobílias, e ao se selecionar uma mobília é exibida a tela de ajuste. Na tela de ajuste é possível transladar e girar a mobília, assim como mudar o eixo em que a mobília está sendo exibida, caso o marcador esteja posicionado em uma parede.

Caso a mobília não agrade o usuário é possível escolher outra mobília ao se selecionar o botão “Change”, que irá direcionar o usuário para tela de seleção de mobílias. Já o botão “Discard” descarta qualquer alteração na tela de ajuste. Ao se clicar no botão “OK” as mudanças são salvas e o usuário voltará para a tela principal.

A qualquer momento o usuário poderá capturar a tela através do botão “Picture” que se encontra na tela principal. Assim a imagem capturada pela câmera com a sobreposição da mobília, porém sem a poluição dos menus do Android, será salva no dispositivo e poderá ser vista na galeria de imagens do Android.

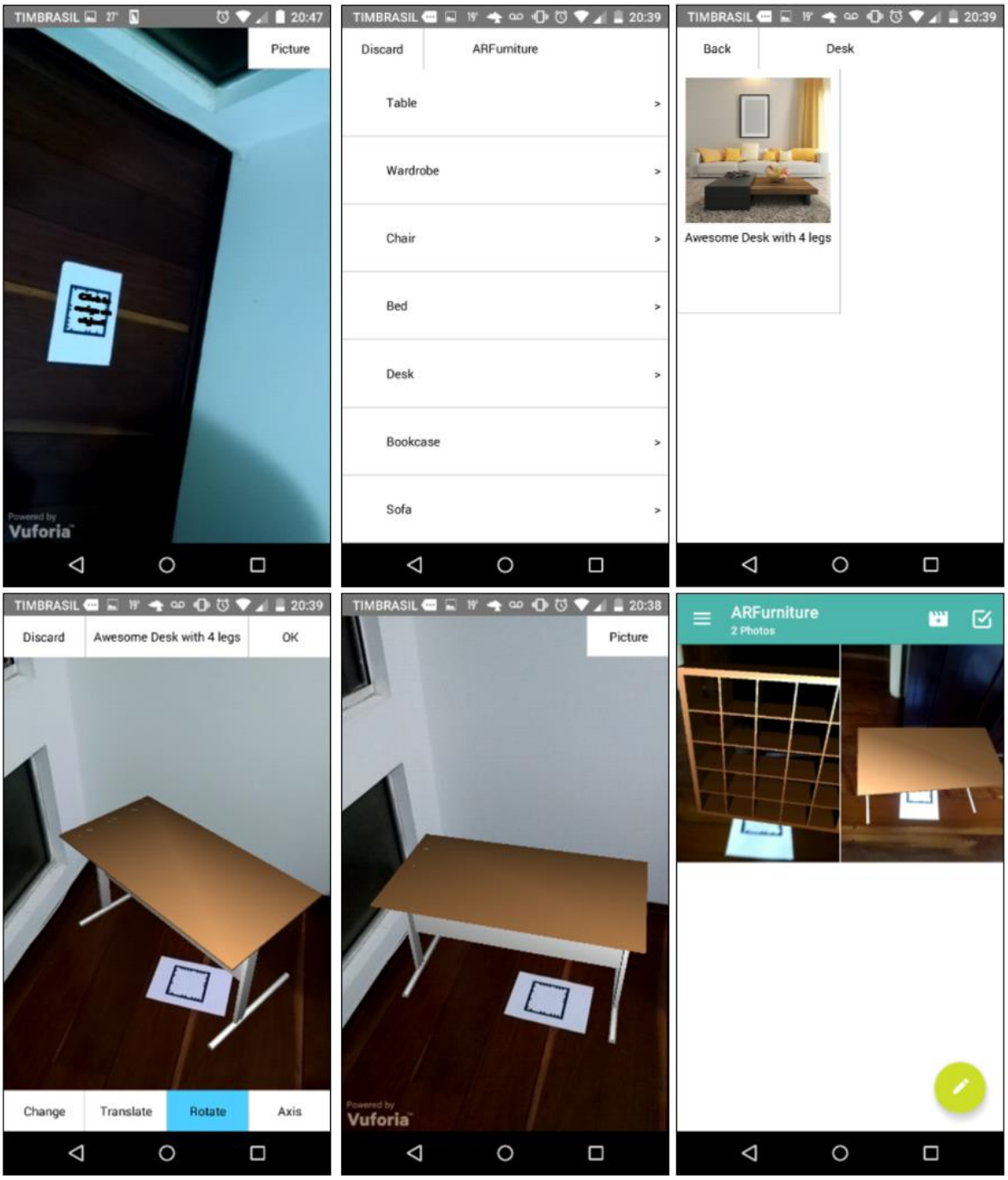

Figura 24: Da esquerda para direita e de cima para baixo: tela principal ao iniciar aplicativo, menu de seleção de categoria, menu de seleção de mobília, tela de ajuste, tela principal contendo a mobília selecionada, galeria de fotos do Android

3.4. Dificuldades e Limitações

Uma das maiores dificuldades encontradas foi justamente a pesquisa e aprendizado sobre ferramentas de suporte a realidade aumentada, assim como a integração delas ao

dispositivo móvel, principalmente pela dificuldade de acesso à documentação e pelas ferramentas muitas vezes estarem desatualizadas com relação a alguma prática nova do Android. Além disso, há a dificuldade natural de se aprender sobre como se utilizar os recursos da maneira certa de forma a melhor aproveitá-los e evitar custos de refatoração futuros.

Outras dificuldades foram encontradas em momentos em que não se seguia o planejamento estipulado pela metodologia XP, como a revisão do código ou as entregas constantes, sendo que essas acarretaram em um custo maior ao se refazer certas partes quando não seguidas. Entretanto, devido ao XP ser muito focado em construções de pequenas partes modulares, isto acarreta em muitas mudanças futuras, assim ressalta-se que mesmo ao se utilizar uma metodologia como essa não se deve descartar uma elaboração da arquitetura geral.

3.5. Considerações Finais

Conclui-se que o desenvolvimento de aplicações de realidade aumentada para dispositivos móveis envolve uma gama de conhecimentos desde algoritmos específicos até metodologias de projetos. Assim, espera-se que o leitor tenha uma noção das dificuldades e resultados obtidos no desenvolvimento por meio das ferramentas e metodologias utilizadas.

Portanto, no próximo capítulo são apresentadas conclusões sobre este trabalho a fim de estabelecer quais objetivos foram cumpridos e o relacionamento deste projeto com a graduação.

CAPÍTULO 4: CONCLUSÃO

4.1. Contribuições

Primeiramente, espera-se que o trabalho contribua para aqueles que gostariam de ter seu primeiro contato desenvolvendo aplicações de realidade aumentada para dispositivos móveis. Assim, o trabalho explana sobre as ferramentas disponíveis de apoio a realidade aumentada, as dificuldades e facilidades sobre as mesmas e a sua integração com o Android. O trabalho também exibe o estado atual das aplicações de RA, dando uma visão geral sobre o mercado e as pesquisas nesta área. Enfim, espera-se que o desenvolvimento ajude aquele que deseja se aventurar em projetos parecidos, exibindo as etapas de planejamento, exemplificando resultados positivos durante o processo e o que deve ser evitado.

De forma semelhante, este trabalho ajudou ao aluno a desenvolver e a tomar conhecimento de um tema antes pouco conhecido por ele, contribuindo com a oportunidade de desenvolver uma aplicação completa em uma área emergente e que há de se tornar cada vez mais comum no futuro. Desta forma, o desenvolvimento da aplicação expande a visão sobre RA assim como permitiu ao aluno praticar fundamentos aprendidos durante a graduação em diversas áreas de conhecimento que a engenharia de computação envolve.

Por fim, espera-se que futuramente a aplicação desenvolvida contribua com um de seus objetivos de longo prazo que é sua participação no mercado por meio da popularização da RA de forma a estar disponível gratuitamente a qualquer um que possua um smartphone, ajudando seus usuários a encontrar diversos tipos de mobílias ou simplesmente como forma de entretenimento e popularização do 3D [7].

4.2. Relacionamento entre o Curso e o Projeto

Devido a diversidade das áreas envolvidas em um projeto como este, o conhecimento adquirido por meio de diversas matérias foi de grande importância para o planejamento e desenvolvimento deste trabalho. Assim, pode-se citar alguns cursos que

auxiliaram diretamente no desenvolvimento da aplicação em si, como a disciplina de Visão Computacional, Computação Gráfica, Engenharia de Software e Sistemas Operacionais.

Da mesma forma algumas disciplinas base ajudaram a formação do conhecimento como grande parte das disciplinas de introdução, como disciplinas de algoritmos e estruturas de dados, os cálculos e álgebras e programação orientada a objetos são alguns exemplos. Disciplinas indiretamente utilizadas, mas que possuem grande importância para projetos futuros que podem ser citadas são Programação Concorrente e Base de Dados.

Além disso, o curso proporcionou a oportunidade de experiência internacional para o aluno e de mercado de trabalho por meio de estágios realizados, assim, expandindo a visão do aluno para além da sala de aula e assim, pode-se colocar em prática também o que se aprendeu com essas experiências no projeto.

Portanto, apesar de destacar-se algumas disciplinas, poder-se-ia citar vários dos cursos cumpridos durante a graduação, de forma que cada um contribuiu para alguma área e expandiu os horizontes do conhecimento, assim, bem servido dentro da ampla área da engenharia de computação o aluno pode escolher a que mais lhe agrada para desenvolver o projeto. Desta forma o curso contribui como o facilitador do acesso ao conhecimento dando base para diversas áreas desconhecidas e liberdade para que se escolha a mais apropriada ao perfil de cada um.

4.3. Trabalhos Futuros

Dentro do planejamento de trabalhos futuros deseja-se polir e sempre buscar melhorias para o projeto até ele atingir maturidade suficiente para ser lançado na loja de aplicativos do Android, mesmo que gratuitamente e como um projeto pessoal. Assim, são propostas futuras:

- O maior número de mobílias e objetos de decoração, o que deve ser consolidado a partir de alguma parceria com artistas ou por meio de objetos que possuem licença gratuita;

- Um sistema *web* para permitir o envio de objetos para que pessoas possam ver suas próprias criações em tamanho real por meio do aplicativo. O sistema também evitaria o sobre carregamento do banco de dados local;

- Um sistema de avaliação e busca para a melhor escolha dos objetos;

Percebe-se, portanto, o foco em um sistema *web*, como um projeto futuro de forma a complementar o aplicativo principal.

REFERÊNCIAS

[1] ALEXANDER, C.; SILVERSTEIN, M.; ISHIKAWA, S. A Pattern Language. Oxford University Press. 190 p. 1977.

[2] ALEXANDER, C. The Timeless Way of Building. Oxford University Press. 552 p. 1979.

[3] ANDROID DEVELOPERS. Android Platform Architecture. Disponível em: < https://developer.android.com/guide/platform/index.html >. Aceso em: 30 out. 2016.

[4] ANDROID DEVELOPERS. Fundamentos de aplicativos. Disponível em: < https://developer.android.com/guide/components/fundamentals.html >. Aceso em: 30 out. 2016.

[5] DAQRI. ARToolKit Documentation. Disponível em: < http://artoolkit.org/documentation/ >. Maio, 2015. Aceso em: 30 out. 2016.

[6] FAUST, F. G. et al. Aplicações e Tendências da Realidade Aumentada no Desenvolvimento de Produtos. In: 8º Congresso Brasileiro de Gestão de Desenvolvimento de Produto, 2011, Porto Alegre.

[7] FERREIRA, R. R. A invasão do mundo 3D, segundo a Microsoft. *Future Behind*. Disponível em: < http://futurebehind.com/windows-10-creators-edition-3d/ >. Outubro, 2016. Aceso em: 30 out. 2016.

[8] GAMMA, E. et al. *Design Patterns*: Elements of Reusable Object-Oriented Software. Estados Unidos, Addison-Wesley, 1995.

[9] GENERAL MOTORS. Gm Reimages Head-Up Display Technology. Março, 2010. Disponível em: http://media.gm.com/content/media/us/en/news/news_detail.brand_gm.html/content/Pages/news/us/en/2010/Mar/0317_hud. Acesso em: outubro, 2016.

[10] GRAHAM, M.; ZOOK, M.; Boulton, A. Augmented reality in urban places: contested content and the duplicity of code. *Transactions of the Institute of British Geographers*, v. 38, n. 3, p. 464-479, 2013.

[11] GRIMM P. et al. Authoring Mixed Reality. In: IEEE International Augmented Reality Toolkit Workshop, n. 1, 2002.

[12] GUIMARAES, G. F. et al. Fpga infrastructure for the development of augmented reality applications. In: SBCCI '07: Proceedings of the 20th annual conference on Integrated circuits and systems design. New York, NY, USA: ACM, 2007, p. 336–341.

[13] HAN, C. Z-x.; Improving Mobile Augmented Reality User Experience on Smartphones. 2010. 39 f. Tese (Master of Science) - Computer Science and Software Engineering, University of Canterbury, Christchurch.

[14] IDC RESEARCH, INC. Smartphone OS Market Share, 2016 Q2. Disponível em: < http://www.idc.com/prodserv/smartphone-os-market-share.jsp >. Agosto, 2016. Aceso em: 30 out. 2016.

[15] JIN, Y-s.; KIM, Y-w; PARK, J. ARMO: Augmented Reality based ReconFigurable Mock-up. In: IEEE and ACM International Symposium on Mixed and Augmented Reality, n. 6, 2007.

[16] KATO, H.; BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality, n. 2, 1999.

[17] KIRNER, C. Evolução da Realidade Virtual no Brasil. In: X Symposium on Virtual and Augmented Reality. João Pessoa, PB, SBC, p. 1-11, 2008.

[18] KIRNER, C.; KIRNER, T. G. Evolução e Tendências da Realidade Virtual e da Realidade Aumentada. *XIII Symposium on Virtual and Augmented Reality*: Livro do Pré-Simposio. Uberlândia: Sociedade Brasileira de Computação, 2011. p. 10-25.

[19] LIBGDX. LibGDX Documentation. Disponível em: < https://github.com/libgdx/libgdx/wiki >. Maio, 2016. Aceso em: 30 out. 2016.

[20] MILGRAM, P. et al. Augmented reality: a class of displays on the reality-virtuality continuum. In: Proceedings of Telemanipulator and Telepresence Technologies, p. 282-292, 1994.

[21] NAIR, S. B.; OEHLKE, A.; Learning LibGDX Game Development. Editora Packt, ed. 2, 2015.

[22] NEEDHAM, M. PTC Completes Acquisition of Vuforia. Disponível em: < http://www.ptc.com/news/2015/ptc-completes-acquisition-of-vuforia >. Novembro, 2015. Acesso em: 30 out. 2016.

[23] NYSTROM, R. Game Programming Patterns. Estados Unidos, Apress, 2011. 300 p.

[24] PTC INC. Vuforia Developer Library. Disponível em: < https://library.vuforia.com/ >. Acesso em: 30 out. 2016.

[25] REIS, B. F. et al. Detecção de Marcadores para Realidade Aumentada em FPGA. *Grupo de Pesquisa em Realidade Virtual e Multimídia Centro de Informática*, Universidade Federal de Pernambuco, 2004.

[26] REIS B. F. et al. Perspective correction implementation for embedded (marker-based) augmented reality. In: Proceedings of Workshop de Realidade Virtual e Aumentada, 2008.

[27] SINGH, M.; SINGH, P. M. Augmented Reality Interfaces. *IEEE Internet Computing*, v. 17, n. 6, p. 66-70, 2013.

[28] STEUER, J. Defining Virtual Reality: Dimensions Determining Telepresence. *Journal of Communication*, v. 42, n. 4, p.73-93, 1993.

[29] SUTHERLAND, I. E. Sketchpad: A man-machine graphical communication system. Computer Laboratory, University of Cambridge. 1963.

[30] SUTHERLAND, I. E. The Ultimate Display. *Proceedings of AFIPS Congress*. pp. 506–508. 1965.

[31] SUTHERLAND, I. E. (1965). A head-mounted three dimensional display. *Proceedings of IFIP Congress*. pp. 9–11. 1968.

[32] THE PLAYROOM. The Playroom Sony PS4. Diponivel em: < https://www.playstation.com/en-us/games/the-playroom-ps4/ >. Acessado em: 30 out. 2016.

[33] WATERS, K. All About Agile. Createspace Independent Publishing Platform, 2012. 382 p.

[34] WILLIAMS, A. et al. An Efficient and Robust Ray-Box Intersection Algorithm. In: ACM SIGGRAPH 2005 Courses, n. 9, p. 49-54, 2005.

[35] XU, L. et al. Intel open source computer vision library, version 3.1. In: Proc. of the 26th Annual Conference on Local Computer Networks, n. 1, 2003.

[36] TOOZLA. Google Play Store. Disponível em: < https://play.google.com/store/apps/details?id=com.toozla.app&hl=en >. Acessado em 24 nov. 2016.

[37] IKEA CATALOGUE. Google Play Store. Disponível em: < https://play.google.com/store/apps/details?id=com.ikea.catalogue.android&hl=en >. Acessado em 24 nov. 2016.

[38] ISTAGING. Google Play Store. Disponível em: < https://play.google.com/store/apps/details?id=com.iStaging.furniture&hl=en >. Acessado em 24 nov. 2016.

[39] PIMENTEL, K.; Teixeira, K. Virtual reality. New York, NY: McGraw-Hill. 1993.

[40] COMEAU, C.; BRYAN, J. Headsight Television System Provides Remote Surveillance. Electronics, pp.86-90. 1961.

[41] LAPLANTE P. A.; NEILL C. J. The Demise of the Waterfall Model Is Imminent and Other Urban Myths. Penn State University. v. 1, n. 10. 2004.

[42] JSON. JSON.ORG. Disponível em: < http://www.json.org >. Acessado em 24 nov. 2016.

[43] SEGAL, M.; AKELEY, K. The OpenGL Graphics System: A Specification. The Khronos Group Inc. 2010.

[44] PLONKA, L. et al. UX Design in Agile: A DSDM Case Study. Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing, p. 1-15. 2014.

[45] COHN, M. Succeeding with Agile: Software Development Using Scrum. Upper Saddle River. Ed. Addison-Wesley. 2010.

[46] MCBREEN, P. Questioning Extreme Programming. Boston, ed. Addison-Wesley. 2003.

[47] DESIGN PATTERNS. Sourcemaking. Disponível em: < https://sourcemaking.com/design_patterns >. Acessado em 24 nov. 2016.