UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

UM ESTUDO SOBRE REALIDADE AUMENTADA PARA A PLATAFORMA ANDROID

GABRIELA TINTI VASSELAI

BLUMENAU
2010

2010/2-14

GABRIELA TINTI VASSELAI

UM ESTUDO SOBRE REALIDADE AUMENTADA PARA A PLATAFORMA ANDROID

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, M. Sc. – Orientador

BLUMENAU
2010

2010/2-14

UM ESTUDO SOBRE REALIDADE AUMENTADA PARA A PLATAFORMA ANDROID

Por

GABRIELA TINTI VASSELAI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Dalton Solano dos Reis, M. Sc. – Orientador, FURB
Membro:	Prof. Paulo Cesar Rodacki Gomes, Dr. – FURB
Membro:	Prof. Francisco Adell Péricas, M. Sc. – FURB

Blumenau, 07 de Dezembro de 2010

Dedico este trabalho à minha família, ao meu namorado e aos meus amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, por iluminar-me durante esta caminhada.

À minha família, pelo apoio e compreensão.

Ao meu namorado, Rafael, pela motivação e paciência.

Aos meus amigos, pelas idéias e dicas.

Ao meu orientador, Dalton Solano dos Reis, pela orientação, direção e dedicação.

Trate as pessoas como se elas fossem o que poderiam ser e você as ajudará a se tornarem aquilo que são capazes de ser.

Goethe

RESUMO

Este trabalho apresenta a implementação de uma aplicação que envolve o conceito de realidade aumentada com geolocalização para dispositivos móveis da plataforma Android. A aplicação desenha pontos de interesse na tela, representados por setas e painéis que direcionam para a localização de cada ponto. Para o desenho de cada ponto de interesse foi utilizada a biblioteca OpenGL ES 1.0 disponível no Android. A interação com a aplicação se dá pela movimentação do dispositivo de forma a acionar a bússola e o acelerômetro. Também a localização do dispositivo determina quão longe os pontos de interesse se encontram. Foram desenvolvidos ainda recursos que permitam ao desenvolvedor utilizar-se de câmera, sensores e coordenadas geográficas na aplicação dentro do simulador do Android. Por fim foi apresentado um estudo de desempenho prático com a aplicação rodando no simulador do Android e também em um dispositivo.

Palavras-chave: Android. Mobilidade. Realidade aumentada.

ABSTRACT

This work describes the construction of an application that evolves the concept of augmented reality with geolocation to mobile devices of Android´s platform. The application draws points of interest on screen, represented by arrows and panels that directs to each point of interest´s location. It was used OpenGL ES 1.0 library available on Android to draw the points of interest. The application´s interaction is made by moving the device in such a way that activates compass and accelerometer. The device´s location determines how far are the points of interest. Were also developed in this work, resources that allow developers to use camera, sensors and geographic coordinates on application inside Android´s simulator. At the end was presented a use case of performance with the application running on Android´s simulator and also on a device.

Key-words: Android. Mobility. Augmented reality.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ADT – *Android Development Tools*

API – *Application Programming Language*

ASCII - *American Standard Code for Information Interchange*

CPU – *Central Processing Unit*

DDMS – Dalvik *Debug Monitor Service*

FPS – *Frames Per Second*

GHz – *Giga Hertz*

GPS – *Global Position System*

GPU – *Graphics Processing Unit*

GPX – GPS *eXchange format*

HTTP – *Hypertext Transfer Protocol*

IP – *Internet Protocol*

JDBC - Java *DataBase Connectivity*

JEE - Java *Enterprise Edition*

JMF – Java *Media Framework*

JNI – Java *Native Interface*

KML – Keyhole *Markup Language*

LAN – *Local Area Network*

MB – *Mega Bytes*

NGA – *National Geospatial-Intelligence Agency*

OpenGL ES – OpenGL for *Embedded Systems*

RAM – *Random Access Memory*

ROM – *Read Only Memory*

UML - *Unified Modeling Language*

URL – *Uniform Resource Locator*

VBO – *Vertex Buffer Object*

WGS – *World Geodetic System*

Wi-Fi – *Wireless Fidelity*

SUMÁRIO

1 INTRODUÇÃO

Durante muitas décadas, a sofisticação das interfaces gráficas computacionais fez com que as pessoas tivessem que se ajustar às máquinas, criando a necessidade de treinamento uma vez que o conhecimento do mundo real já não era suficiente. Com a evolução das tecnologias de hardware e software foi possível fazer com que as máquinas se ajustassem às pessoas, possibilitando aos usuários acessarem aplicações como se estivessem atuando no mundo real, falando, apertando ou fazendo gestos (KIRNER; SICOUTTO, 2007, p. 3).

Neste contexto, surge na década de 90 a realidade aumentada que consiste em suplementar o mundo real com objetos virtuais sobrepondo ou combinando os objetos reais (AZUMA, 1997, p. 257), desta forma permitindo uma interação homem-máquina mais fácil e natural. Azuma (1997, p. 257) considera ainda que um sistema de realidade aumentada deve possuir três características: combinar objetos reais e objetos virtuais em um ambiente real, executar interativamente em tempo real e alinhar objetos reais e virtuais entre si.

Um dos desafios computacionais da realidade aumentada está em utilizar dispositivos de hardware que permitam a entrada, o processamento e a saída do vídeo oferecendo o mínimo de tempo de resposta possível para que seja mantida a noção de realidade. Bimber e Raskar (2005, p. 71) classificam estes dispositivos de hardware em *head-attached display* (dispositivo acoplado a cabeça), *spatial display* (dispositivo espacial) e *hand-held display* (dispositivo móvel). Desses, o dispositivo móvel é o único que alinha popularidade e baixo custo com a capacidade mínima necessária para o processamento de realidade aumentada.

Os dispositivos móveis têm evoluído de forma notável nos últimos tempos tanto na sua capacidade de processamento, armazenamento como também na quantidade de funcionalidades disponíveis. Segundo Schemberger, Freitas e Vani (2009), essa evolução foi motivada pelo rápido crescimento no número de consumidores, passando de dois bilhões de usuários em 2005 para quatro bilhões em 2008. Diante deste cenário, surge a plataforma Android, para dispositivos móveis, tendo como base um sistema operacional Linux e disponibilizando bibliotecas para desenvolvimento na linguagem Java (GOOGLE, 2010p). Android destaca-se não só pelas suas funcionalidades de multitoque, bússola digital e *Global Position System* (GPS), mas também por ser a primeira plataforma para aplicações móveis, completamente livre e de código aberto (SCHEMBERGER; FREITAS; VANI, 2009).

Com o exposto acima, este trabalho pretende estudar o potencial da plataforma Android no que diz respeito ao desenvolvimento de realidade aumentada através de uma

aplicação que utiliza os recursos disponíveis na plataforma, como a câmera de vídeo, o acelerômetro e os fornecedores de localização geográfica.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma aplicação que envolva realidade aumentada na plataforma Android.

Os objetivos específicos do trabalho são:

a) elucidar o potencial da plataforma Android frente ao conceito de realidade aumentada;
b) aplicar o conceito de realidade aumentada que utiliza o registro dos objetos virtuais através de coordenadas geográficas;
c) utilizar os recursos de câmera de vídeo, GPS, acelerômetro e bússola disponibilizados na plataforma Android, para uma melhor interação do usuário com a realidade aumentada.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em quatro capítulos, sendo que o segundo capítulo contém a fundamentação teórica necessária para o entendimento deste trabalho.

O terceiro capítulo apresenta como foi desenvolvida a aplicação na plataforma Android, os casos de uso da aplicação, os diagramas de classe e toda a especificação que define a aplicação. Ainda no terceiro capítulo são apresentadas as partes principais da implementação e também os resultados e discussões que aconteceram durante toda a etapa de desenvolvimento do trabalho.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é descrita a realidade aumentada e suas características. Na seção 2.2 é apresentada a plataforma Android e suas funcionalidades. Por fim, a seção 2.3 traz três trabalhos correlatos.

2.1 REALIDADE AUMENTADA

Bimber e Raskar (2005, p. 2) definem a realidade aumentada, de maneira bem genérica, como sendo uma forma de integrar informação sintética dentro de um ambiente real. Esta informação sintética pode ser uma imagem, um som ou até mesmo uma simulação de toque. Neste sentido, aumentar a realidade é somar ao mundo real alguma informação virtual de maneira a complementá-lo, permitindo alguma interação entre o usuário e o ambiente gerado.

De acordo com Azuma (1997, p. 2) existem três propriedades básicas que definem um sistema que implementa o conceito de realidade aumentada. Estas propriedades estão inter-relacionadas com os elementos fundamentais descritos por Bimber e Raskar (2005, p. 4).

A primeira propriedade descrita por Azuma (1997, p. 2) trata da combinação da realidade com a virtualidade. Para essa combinação é preciso o uso de alguma tecnologia de hardware específica. Dispositivos do tipo *head-mounted* (acoplados a cabeça) são a tecnologia dominante para aplicações de realidade aumentada (BIMBER; RASKAR, 2005, p. 5). No entanto, esses dispositivos ainda possuem grandes limitações ópticas (limitando o campo e o foco de visão), técnicas (resolução limitada e a instabilidade das imagens) e do fator humano (pelo elevado tamanho e peso do dispositivo).

Os dispositivos do tipo *hand-held* (móveis) permitem ao usuário maior controle do espaço que terá a sua realidade aumentada, uma vez que pode movimentá-lo para qualquer direção. Nesse tipo de dispositivo o conceito mais usado é o de *video see-through* (visão através de vídeo), no qual a câmera captura o vídeo em tempo real que é sobreposto por objetos gráficos antes de ser visualizado pelo usuário (BIMBER; RASKAR, 2005, p. 79).

Alguns sistemas de realidade aumentada não necessitam de mobilidade. Para estes casos pode ser usado o *spatial display* (dispositivo espacial). Este tipo de dispositivo é

totalmente desacoplado ao usuário e integrado ao ambiente. Dentre as categorias de dispositivo espacial pode-se citar as baseadas em projeções. Nesta, segundo Calife (2008, p. 26), o conteúdo virtual é projetado diretamente na superfície dos objetos físicos, sendo compostos de um ou mais projetores, dependendo do espaço físico que será projetado.

O segundo grande pilar da realidade aumentada é o registro e rastreamento dos objetos em terceira dimensão (AZUMA, 1997, p. 2). Esse é um dos desafios fundamentais das pesquisas de realidade aumentada nos dias de hoje (BIMBER; RASKAR, 2005, p. 5). Com relação ao rastreamento, o sistema precisa determinar de maneira rápida, precisa e robusta o posicionamento do usuário, seu ponto de vista e a posição dos objetos reais e virtuais no ambiente (CALIFE, 2008, p. 23).

Para o registro da posição dos objetos virtuais no ambiente real são utilizadas algumas estratégias, dentre as quais pode-se destacar o uso de marcações em forma de códigos impressos nos objetos reais. Essas marcações são interpretadas pelo dispositivo que passa a mostrar o objeto virtual correspondente à marcação (ARTAG, 2010). Outra estratégia, que é comumente usada em dispositivos com baixa capacidade de processamento de imagens, é a de geolocalização. Nessa, os objetos virtuais são registrados em determinada posição geográfica e através do GPS e do acelerômetro é possível medir aonde o objeto virtual deve ser mostrado (BIMBER; RASKAR, 2005, p. 5).

A última propriedade, segundo Azuma (1997, p. 2), é a de interatividade em tempo real. Uma vez que a realidade aumentada concentra-se em sobrepor o ambiente real por elementos gráficos, métodos para renderização rápida e realística têm um importante papel. Mesmo renderizados em alta qualidade visual, os objetos virtuais devem ser integrados no ambiente real de forma consistente, incorporando as sombras do ambiente e o comportamento de inter-reflexão dos objetos (BIMBER; RASKAR, 2005, p. 5).

2.2 ANDROID

Android é uma plataforma de software para dispositivos móveis que inclui um sistema operacional, uma camada intermediária e aplicações chave (GOOGLE, 2010p). Dentre os diversos recursos disponíveis no Android pode-se citar o *framework* para desenvolvimento de aplicativos, a máquina virtual Dalvik e o suporte a recursos de multimídia (áudio, imagem e vídeo).

2.2.1 Arquitetura do sistema operacional

Segundo Google (2010p), a arquitetura do Android é dividida em quatro camadas e cada uma desempenha uma função particular.

A primeira camada tem a função de abstrair os recursos de hardware para as demais camadas. Ela é composta pelo *kernel* Linux 2.6 que executa as tarefas de gerenciamento de memória, gerenciamento de processos e gerenciamento de rede (GOOGLE, 2010p).

Composta por bibliotecas desenvolvidas na linguagem C++, a segunda camada serve como base para as aplicações, fornecendo diversos recursos que entre eles pode-se destacar uma base relacional leve chamada `SQLite`, uma implementação baseada na OpenGL for *Embedded Systems* (OpenGL ES) na versão 1.0, suporte à criação e reprodução de vários formatos de mídias e um renderizador de fontes *Bitmap* ou vetoriais (GOOGLE, 2010p). Ainda na segunda camada existe a máquina virtual Dalvik, que gerencia a execução dos aplicativos de forma a garantir um baixo consumo de memória.

A terceira camada é composta por uma *Application Programming Language* (API) desenvolvida em Java para abstrair o uso das bibliotecas da segunda camada. Esta API invoca as bibliotecas da segunda camada através das interfaces Java *Native Interface*[1] (JNI), tornando simples o reuso dos componentes e permitindo aos desenvolvedores o acesso aos mesmos recursos utilizados pelas aplicações do sistema.

Na quarta e última camada estão os aplicativos disponíveis na plataforma, como cliente de *e-mail*, calendário, mapa e contatos. Estes aplicativos reusam os recursos disponibilizados na terceira camada (GOOGLE, 2010p).

2.2.2 Máquina virtual Dalvik

A máquina virtual Dalvik foi desenvolvida para executar as aplicações em um dispositivo com baixo poder de processamento, pouca memória física, pouca *Random Access Memory* (RAM) e que tenha como sua fonte de energia a bateria (GOOGLE, 2010b).

Para que as classes Java pudessem ser compiladas e executadas em tal dispositivo, foi

[1] Java *Native Interface* é um padrão de programação de interface para escrever métodos nativos Java e embutir a máquina virtual Java em aplicações nativas. O objetivo principal é a compatibilidade binária de bibliotecas nativas com todas as implementações de máquinas vituais Java de determinada plataforma (ORACLE, 2010).

necessário rever a geração dos *bytecodes* Java e os arquivos com extensão `class`[2] (GOOGLE, 2010b). Segundo Google (2010b), há muita redundância entre esses arquivos e para evitar tal excesso foi criado o arquivo com a extensão `dex` (Dalvik *Executable*). Um arquivo com a extensão `dex` equivale a um ou mais arquivos com a extensão `class` otimizados e ocupa na memória física do dispositivo 45% do espaço que um arquivo com a extensão `jar` não comprimido ocuparia (GOOGLE, 2010b).

Um dos processos mais importantes executados dentro do sistema operacional Android é chamado Zygote (GOOGLE, 2010b). Este processo é iniciado junto com o sistema operacional e sua responsabilidade é gerenciar a memória e o ciclo de vida dos aplicativos, bem como pré-carregar as classes que serão usadas em vários aplicativos (GOOGLE, 2010b). Segundo Google (2010b), para cada aplicativo aberto é criada uma instância da máquina virtual Dalvik e o responsável por gerenciar as diversas máquinas virtuais abertas é o próprio processo Zygote.

Para evitar que na memória fiquem objetos não mais utilizados, a máquina virtual Dalvik possui um *garbage collector*. Segundo Google (2010b), existem duas estratégias para gerenciamento das informações relativas ao uso dos objetos na memória. A primeira delas consiste em guardar as informações de uso de um objeto junto com o próprio espaço de memória reservado para o objeto. A segunda estratégia reserva um espaço da memória para guardar as informações relativas ao uso de todos os objetos carregados. A estratégia usada no Android foi a segunda, por ser a mais otimizada em termos de espaço ocupado (GOOGLE, 2010b).

2.2.3 Intenções

Android possui um inovador mecanismo de troca de mensagens através de intenções. Uma intenção, representada pela classe `Intent`, consiste em uma funcionalidade que uma aplicação requisita de outras, indicando a intenção de reuso (GOOGLE, 2010f). Segundo Google (2010f), uma vez requisitada a intenção, o Android encarrega-se de procurar o aplicativo mais apropriado para responder à requisição e, caso necessário, cria uma instância deste novo aplicativo.

---

[2] Arquivo que contém instruções de máquina independentes de plataforma que serão executadas pela máquina virtual Java (SUN MICROSYSTEMS, 2010).

Para cada intenção requisitada é preciso informar uma ação, que representa a funcionalidade a ser atendida, e informações complementares, que representam os parâmetros necessários para atender a funcionalidade (GOOGLE, 2010f). Um exemplo de aplicação que disponibiliza intenção é o navegador, sendo `view` a ação utilizada para requisitá-lo e é necessário passar a ele a *Uniform Resource Locator* (URL) que será aberta. Outras aplicações podem registrar-se no Android como aplicações que disponibilizam a ação `view`. Quando esta intenção for requisitada, a plataforma encarrega-se de decidir qual navegador é o mais apropriado. Através deste mecanismo é possível modularizar as funcionalidades e aumentar a reutilização de recursos da plataforma (GOOGLE, 2010f).

2.2.4 Recursos de multimídia

A plataforma Android fornece recursos de codificação/decodificação de uma variedade de tipos comuns de mídia, de modo a permitir que facilmente sejam integrados áudio, imagem e vídeos nos aplicativos (GOOGLE, 2010a). É possível reproduzir áudio e vídeo de vários tipos de fontes de dados no Android. Segundo Google (2010a), estes recursos podem ser mídias armazenadas no aplicativo, mídias armazenadas no sistema de arquivos ou até mesmo de um fluxo de dados vindo de uma conexão de rede. A plataforma também permite a gravação de áudio e vídeo através da classe `MediaRecorder`. Essa funcionalidade é bastante dependente da disponibilidade do hardware do dispositivo e não é possível usá-la pelo simulador (GOOGLE, 2010a).

Para o desenho de interface gráfica com o usuário, o Android possui uma API com bastantes componentes prontos como botões, campos textuais, botões de marcação, botões do tipo radio, menus, entre outros (GOOGLE, 2010o). Já no que diz respeito a desenho e renderização de imagens 2D há duas opções disponíveis (GOOGLE, 2010d). A primeira opção diz respeito ao desenho de imagens fixas que não serão mudadas dinamicamente, onde a implementação pode utilizar-se da classe `View`. Essa é considerada a melhor opção por utilizar os recursos normais de pintura do sistema. A segunda opção deve ser usada quando a aplicação precisa controlar a animação de pintura das imagens e sua implementação é feita sobrescrevendo métodos da classe `Canvas` (GOOGLE, 2010d).

O Android também possui suporte a OpenGL ES 1.0 que pode ser usado tanto para renderização de objetos em 2D quanto para 3D (GOOGLE, 2010d).

2.2.5 OpenGL ES

OpenGL ES é uma subseção da API da biblioteca de gráficos tridimensionais OpenGL, mantida pelo Khronos Group, projetada para sistemas embarcados como telefones celulares e *consoles* de vídeo games (WIKIPEDIA, 2010d). A versão 1.0 da OpenGL ES foi projetada de acordo com a versão 1.3 da OpenGL. Segundo Wikipedia (2010d), na criação da especificação da OpenGL ES muitas funcionalidades presentes na API original da OpenGL foram removidas e algumas poucas foram adicionadas. A maior diferença entre a OpenGL e a OpenGL ES está na remoção da chamada `glBegin` e `glEnd`, tornando possível o desenho de primitivas apenas com o uso de vetores de vértices (WIKIPEDIA, 2010d). A segunda maior diferença está na introdução do tipo numérico de ponto fixo para a coordenada dos vértices e atributos, visando melhor suporte aos sistemas embarcados que não suportam o tipo numérico de ponto flutuante (WIKIPEDIA, 2010d).

Os aplicativos que usam a OpenGL ES no Android, chamam os métodos da interface `GL10` que possui todas as funcionalidades da especificação 1.0 da OpenGL ES. A interface acima denominada faz chamadas nativas via JNI para as bibliotecas desenvolvidas em C/C++ que implementam o padrão OpenGL ES (GRAPH TECH, 2010).

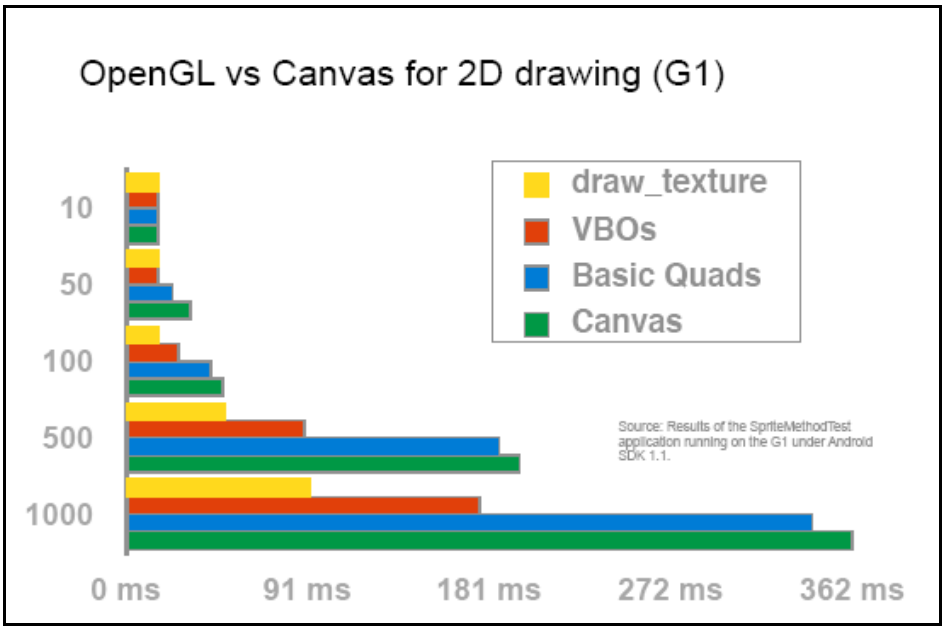
Testes de desempenho feitos pela Google (2010q) dentro do dispositivo G1[3] comparam o uso de Canvas e OpenGL na renderização de uma quantidade grande de imagens 2D, conforme mostra a Figura 1.

Foram comparadas quatro funcionalidades diferentes de renderização de objetos 2D, as três primeiras fazem uso de OpenGL ES e a quarta é implementada através da classe `Canvas`. A comparação se dá na quantidade de objetos (representada na vertical) pelo tempo em milissegundos (representado na horizontal).

A primeira funcionalidade utiliza a extensão `GL_OES_draw_texture` para desenhar texturas na tela sem nenhuma ordem de projeção ou vetores de vértices envolvidos (GOOGLE, 2010l). Essa funcionalidade, segundo Google (2010l), deve ser usada com cautela, pois pode não ser suportada em todos os dispositivos que possuem o Android como plataforma. Para verificar se há suporte a esta funcionalidade deve-se chamar o método `glGetString(GL_EXTENSIONS)` dentro da interface `GL10` (GOOGLE, 2010l).

---

[3] G1 ou Android Dev Phone 1 são denominações usadas para o dispositivo desenvolvido pela Google em parceria com a HTC para fornecer um celular compatível com a plataforma Android. Trata-se de uma versão do modelo HTC Dream desbloqueado para ser usado apenas por desenvolvedores (WIKIPEDIA, 2010a)



Fonte: Linux Graphics (2010).

Figura 1 - Comparação entre OpenGL e Canvas

A segunda funcionalidade demonstrada na Figura 1 utiliza vetores de vértices para descrever as texturas na forma de *buffers* de vértices ou *Vertex Buffer Object* (VBO), tornando a memória da *Graphics Processing Unit* (GPU) responsável por armazenar os objetos (GOOGLE, 2010l). Assim como a funcionalidade anterior, esta pode não ser suportada por todos os dispositivos e antes de usá-la deve-se fazer a mesma verificação demonstrada acima.

Segundo Google (2010l) a terceira funcionalidade pinta as texturas em um plano com vértices básicos usando a memória principal do dispositivo e com projeção ortográfica. Como essa funcionalidade requer apenas os recursos básicos da OpenGL ES 1.0, há a garantia de ser suportada por todos os dispositivos (GOOGLE, 2010l).

A última funcionalidade pinta as texturas usando objetos de Bitmap em um objeto do tipo `Canvas`. Essa é uma funcionalidade que usa apenas operações na *Central Processing Unit* (CPU) e também é a mais simples de implementar entre as quatro demonstradas (GOOGLE, 2010l).

Cada teste utilizou uma quantidade diferente de objetos, variando de 10 a 1000 objetos renderizados simultaneamente dentro do dispositivo. O teste de cada funcionalidade foi executado em separado e dentro do mesmo dispositivo, o G1 (GOOGLE, 2010l).

A partir desses testes pode-se concluir, segundo Google (2010l), que a classe `Canvas` é fácil de implementar mas o seu desempenho está muito abaixo do esperado. Embora não foram testadas a rotação e a escala de texturas, seu uso em `Canvas` seria muito caro. O uso de OpenGL ES demonstrou ser a melhor estratégia para aplicações em 2D que precisam manter uma alta taxa de atualização de quadros ou quando há o uso de rotação e escala das texturas.

Ainda segundo Google (2010l), as extensões da OpenGL ES demonstradas, em particular a `GL_OES_draw_texture`, produziram bons resultados no dispositivo G1. Todavia nem todos os dispositivos possuem essas extensões, portanto o código deve estar preparado para tratar dispositivos que não as suportam. Também o uso de vetores de vértices estáticos (tanto em 2D quanto em 3D) no dispositivo G1 torna o VBO uma ótima estratégia (GOOGLE, 2010l).

A chamada excessiva às APIs nativas via JNI deve ser levada em consideração quando usar OpenGL ES. Cada chamada de método na interface `GL10` invocará através do JNI a implementação C++ correspondente e após certo tempo tal excesso pode tornar-se considerável. Segundo Google (2010l), a melhor estratégia seria chamar métodos de pintura que são mais lentos quando usados individualmente, mas que resultam em menos chamadas nativas; por exemplo, um jogo baseado em texturas provavelmente é mais rápido quando renderizado com vetor de vértices usando a extensão *Vertex Buffer Object* (VBO) do que usando chamadas individuais de pintura para cada textura. Através desses testes pode-se concluir que a pintura de textura é a maneira mais rápida de pintar texturas 2D no G1, mas cada chamada de pintura requer pelo menos uma chamada nativa JNI, neste caso um vetor de vértices torna-se mais rápido (GOOGLE, 2010l).

2.2.6 Serviços de localização

Android possui classes que permitem que aplicações tenham acesso aos serviços de localização geográfica suportados pelos dispositivos (GOOGLE, 2010g). Segundo Google (2010g), o componente central para o *framework* de localização é a classe `LocationManager` que fornece APIs para determinar a localização e direção do movimento.

De acordo com Google (2010g), assim como outros serviços do sistema Android, a classe `LocationManager` não é instanciada diretamente pela aplicação que irá usá-la. É necessário requisitar ao sistema a sua instância através da chamada `getSystemService(Context.LOCATION_SERVICE)`. Uma vez com a instância dessa classe é possível obter a lista de todos os fornecedores de localização disponíveis (representados pela classe `LocationProvider`). Também é possível registrar-se em determinado fornecedor de localização para receber atualizações periódicas da mudança de localização do dispositivo ou quando o dispositivo aproxima-se de determinada latitude e longitude (GOOGLE, 2010g).

Os dois principais fornecedores de localização no Android são a rede e o GPS. Apesar

de ser mais preciso o GPS funciona apenas em ambientes abertos, consome rapidamente a energia da bateria e não retorna a localização de forma tão rápida quanto o usuário espera (GOOGLE, 2010i). Por outro lado o fornecedor de localização que utiliza a rede determina a localização através da torre que se comunica com o celular e de sinais do tipo *Wireless Fidelity* (Wi-Fi) de forma a funcionar tanto em ambientes abertos quanto fechados, sua resposta é mais rápida e a bateria é menos utilizada do que no GPS (GOOGLE, 2010i).

O registro para obtenção da localização obedece ao padrão de projeto *Observer* [4]. No Android a classe observadora de mudanças na localização deve implementar a interface `LocationListener` e esta deve ser registrada na classe `LocationManager` através do método `requestLocationUpdates(String,long,float,LocationListener)` (GOOGLE, 2010i). O primeiro parâmetro do método `requestLocationUpdates` é o tipo de fornecedor de localização que será utilizado. Os tipos de fornecedores rede e GPS podem ser obtidos através das constantes `LocationManager.NETWORK_PROVIDER` e `LocationManager.GPS_PROVIDER` respectivamente. O segundo e terceiro parâmetros controlam a freqüência com que a aplicação deseja ser atualizada, sendo o segundo parâmetro o intervalo de tempo mínimo em milissegundos e o terceiro parâmetro a distância mínima em metros (GOOGLE, 2010i). Se ambos o segundo e o terceiro parâmetros forem passados como zero, a freqüência de atualização será a máxima possível. O último parâmetro é o objeto observador que irá receber todas as atualizações de mudança na localização (GOOGLE, 2010i).

De acordo com Google (2010i), obter uma localização confiável pode se tornar uma tarefa complicada, pois os fornecedores de localização podem conter erros ou serem imprecisos. Cada fornecedor de localização pode fornecer uma localização diferente, determinar qual será utilizado é uma questão de escolher entre precisão, velocidade e eficiência da bateria (GOOGLE, 2010i). Ainda segundo Google (2010i), outro fator considerável é a movimentação do usuário que deve ser capturada com freqüência para manter a aplicação atualizada. O último fator de risco, segundo Google (2010i) é a estimativa de acordo com a precisão da fonte de localização, a localização atual pode ser mais precisa que a obtida dez segundos antes, mesmo estas sendo oriundas da mesma fonte de localização.

Para manter a eficiência da bateria do dispositivo, Google (2010i) recomenda que as aplicações fiquem o tempo mínimo possível observando as mudanças de localização de forma

[4] *Observer* é um padrão de projeto de software que define dependência entre objetos de modo que quando o objeto observado muda o estado, todos os seus dependentes devem ser notificados e atualizados automaticamente. Permite que os objetos interessados sejam avisados da mudança de estado ou outros eventos ocorridos no objeto observado (WIKIPEDIA, 2010c).

a registrarem-se e desregistrarem-se na classe `LocationManager` quantas vezes for necessário.

Quando a aplicação necessita receber a localização o mais rápido possível, Google (2010i) recomenda que seja utilizado o método `getLastKnownLocation` da classe `LocationManager` passando como parâmetro o tipo de fornecedor. Esta localização pode estar desatualizada quando, por exemplo, o dispositivo for desligado e movido para outra localização (GOOGLE, 2010h).

Aplicações que necessitam a localização de forma precisa devem testar os valores obtidos, pois a localização mais recente pode não ser a de maior precisão (GOOGLE, 2010i). Três são as validações recomendadas pela Google (2010i) para manter a melhor estimativa da localização. A primeira delas é verificar se a nova localização obtida é significativamente mais nova do que a anterior, a variação de tempo usada depende de cada aplicação. A segunda validação é sobre a precisão da localização recebida com relação à última localização considerada válida. A terceira trata do nível de confiança que a aplicação deposita no fornecedor de localização (GOOGLE, 2010i).

Fornecer o serviço de localização é responsabilidade, principalmente, do hardware do dispositivo. Como esse serviço não está disponível no simulador do framework de desenvolvimento do Android, pode-se utilizar três maneiras de simular a localização geográfica.

A primeira maneira é através da ferramenta de depuração e monitoramento de aplicativos para Android chamada Dalvik Debug *Monitor Service* (DDMS), é possível importar arquivos de formato GPS *eXchange format* (GPX)[5] ou Keyhole *Markup Language* (KML)[6] que simulam a movimentação geográfica (GOOGLE, 2010i).

A segunda maneira é através de linha de comando aonde apenas uma coordenada pode ser informada por vez. Através do terminal é possível conectar-se ao simulador via *telnet* no comando `telnet localhost <porta_do_simulador>`. Uma vez conectado, executar o comando `geo fix <latitude> <longitude> <altitude>` para determinar uma coordenada fixa (GOOGLE, 2010i).

Na implementação da aplicação também é possível simular a localização geográfica. Através do método `addTestProvider` da classe `LocationManager` é possível adicionar-se

[5] O formato GPX é baseado em XML e foi criado para unificar informações oriundas de GPS como pontos de trajeto, rotas e trilhas (TOPOGRAFIX, 2010).

[6] O formato KML é baseado em XML e foi criado para expressar anotações geográficas em 2D e em 3D. Esse formato foi desenvolvido inicialmente para uso com o Google Earth (WIKIPEDIA, 2010b).

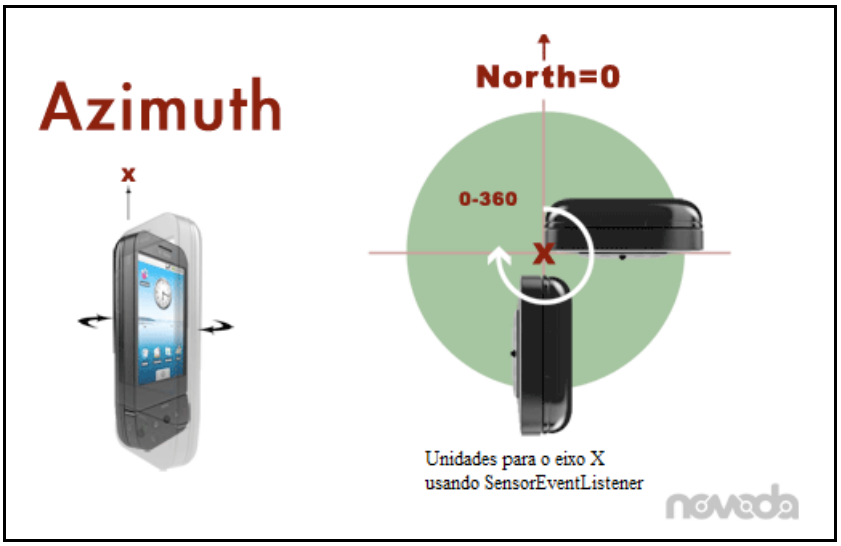
como fornecedor de localizações geográficas para testes (GOOGLE, 2010h). Para avisar que houve uma mudança de localização deve-se chamar o método `setTestProviderLocation`, da classe `LocationManager`, passando um objeto de localização da classe `Location` (GOOGLE, 2010h).

2.2.7 Sensores

Android tem uma API que suporta uma variedade grande de sensores, tais como o acelerômetro ou sensor de luz (GOOGLE, 2010j). Os sensores mais comumente usados são os acelerômetros e os sensores magnéticos (ou bússolas). As aplicações usam esses sensores como forma de entrada de dados vinda do usuário e para orientar a tela (GOOGLE, 2010j).

Assim como no serviço de localização, os sensores no Android disponibilizam seus dados através do padrão de projeto *Observer*. A classe observadora de mudanças nos sensores deve implementar a interface `SensorEventListener` e deve registrar-se como observador através da classe `SensorManager` no método `registerListener(SensorEventListener, Sensor, int)` (GOOGLE, 2010k). O primeiro parâmetro do método deve ser o observador e o terceiro parâmetro trata da freqüência com que o observador quer ser avisado sobre as mudanças (GOOGLE, 2010k). O segundo parâmetro será o sensor observado podendo ser obtido pelo método `SensorManager.getSensorList(int)` que retorna uma lista de sensores para determinado tipo, ou através do método `SensorManager.getDefaultSensor(int type)` que retorna o principal sensor para determinado tipo (GOOGLE, 2010k). O método que recebe as mudanças de sensores é `onSensorChanged` e através do seu único parâmetro são obtidos os valores do sensor no atributo `SensorEvent.values`, a precisão no atributo `SensorEvent.accuracy`, o próprio sensor no atributo `SensorEvent.sensor` e o tempo das medições no atributo `SensorEvent.timestamp` (GOOGLE, 2010k)

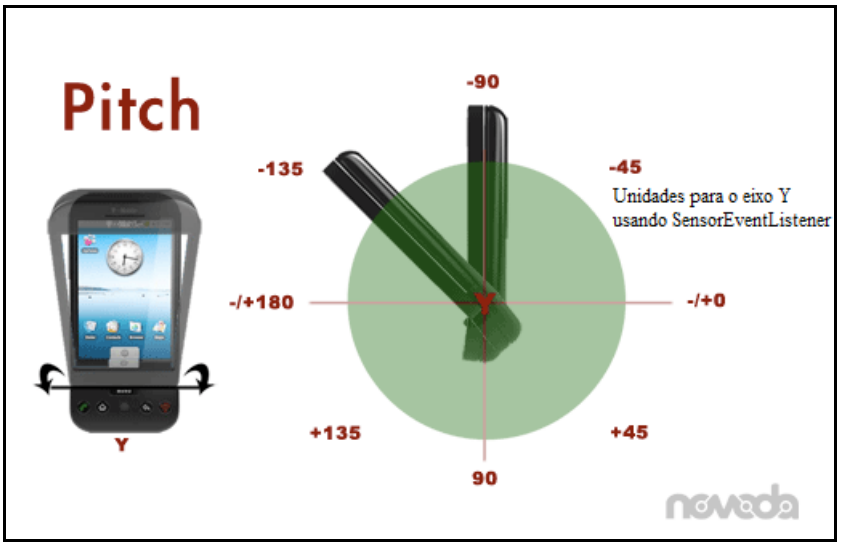
Quando o sensor utilizado é o de orientação, três são os valores principais fornecidos ao ocorrer alguma mudança no dispositivo. O primeiro valor chama-se *azimuth* e trata do ângulo de rotação do dispositivo no eixo X (NOVODA, 2010). Esse ângulo tem por referência no ponto 0º o Norte Magnético da Terra e suas unidades variam de 0º a 359º (NOVODA, 2010). A Figura 2 representa a movimentação do dispositivo em *azimuth*.



Fonte: Novoda (2010).

Figura 2 - Movimentação do dispositivo no eixo X

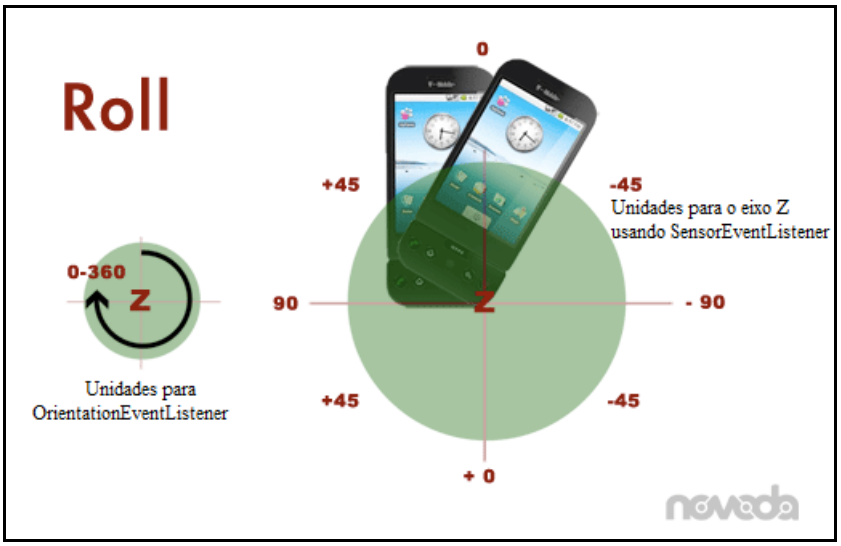
O segundo valor de orientação obtido chama-se *pitch* e sua rotação incide no eixo Y (NOVODA, 2010). Quando o dispositivo está em sua posição padrão, o valor de *pitch* será 90º negativos, na medida em que o dispositivo for rotacionado de forma a aproximar o visor do chão ou a mostrar o visor para o céu, o valor de *pitch* varia para 180º ou 0º respectivamente (NOVODA, 2010). A Figura 3 representa a movimentação do dispositivo em *pitch*.



Fonte: Novoda (2010).

Figura 3 – Movimentação do dispositivo no eixo Y

O terceiro valor de orientação chama-se *roll* e sua rotação incide no eixo Z (NOVODA, 2010). Ainda segundo Novoda (2010), esse valor determina a rotação do dispositivo em relação ao canto esquerdo inferior da tela e está representado na Figura 4.



Fonte: Novoda (2010).

Figura 4 - Movimentação do dispositivo no eixo Z

Existem dois tipos de orientação da tela no Android, a primeira delas chama-se *portrait* e é obtida quando o dispositivo está mais alto do que largo. A segunda chama-se *landscape* e é obtida quando o dispositivo está mais largo do que alto. Uma aplicação que atualiza a tela com base nos sensores de orientação deve considerar diferentes cálculos para as duas formas possíveis de orientação do dispositivo (GOOGLE, 2010j).

Para a correta utilização da API de sensores é necessário seguir três regras básicas, segundo Google (2010j):

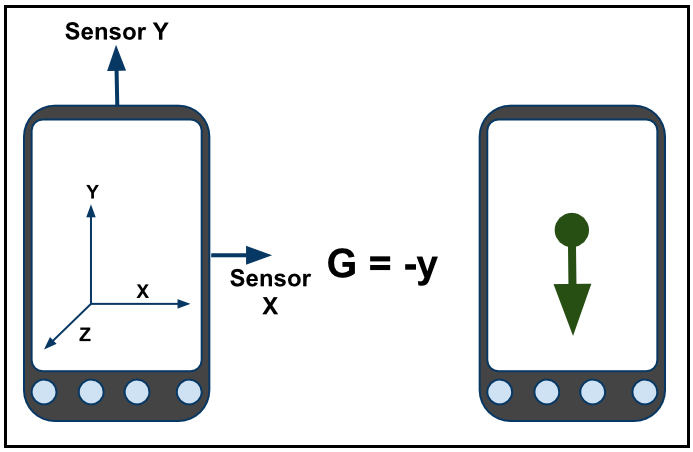
a) o sistema de coordenadas dos sensores para a orientação natural do dispositivo não muda na medida que o dispositivo é movido, e é o mesmo sistema de coordenadas usado na OpenGL ES;

b) as aplicações não podem assumir que a orientação natural de todos os dispositivos é *portrait*, quão menos *landscape*;

c) aplicações que tratam os dados dos sensores para alinhar a tela, devem sempre usar o método `Display.getRotation()` para converter as coordenadas do sensor para as coordenadas da tela (GOOGLE, 2010j).

Considera-se um exemplo de aplicação que desenha uma seta para a direção da gravidade, animando a seta quando o dispositivo é movido, como um fio de prumo[7]. Essa aplicação, quando o dispositivo estiver na sua orientação padrão, será desenhada conforme representado pela Figura 5.

[7] Fio de prumo é um instrumento utilizado na topografia para a medição de distâncias, é composto por um cordel que segura um peso que determina a posição vertical com exatidão (WIKIPEDIA, 2010e).

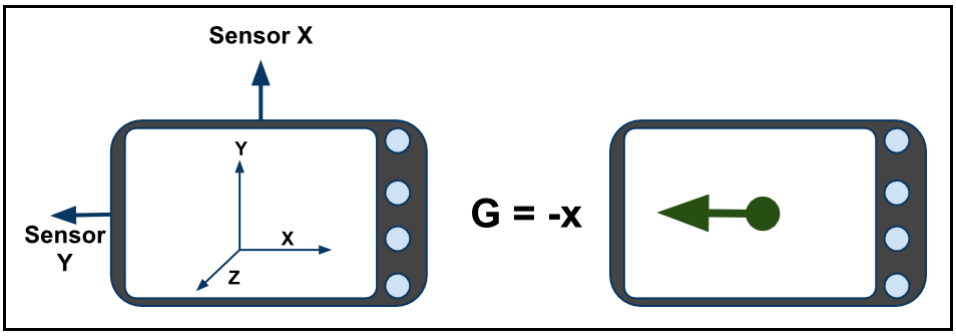


Fonte: Google (2010j).

Figura 5 - Sistema de coordenadas quando o dispositivo está em sua orientação padrão

De acordo com Google (2010j), quando o dispositivo está em sua orientação padrão, a gravidade (representada pela letra G) é alinhada com o eixo Y na direção negativa. Essa aplicação pode ser facilmente implementada em OpenGL ES, uma vez que os sistemas de coordenadas são os mesmos, desenha-se uma seta e a rotação é feita pelos dados do acelerômetro (GOOGLE, 2010j).

Muitos dispositivos utilizam o acelerômetro para detectar sua rotação e a tela é rotacionada de acordo, automaticamente (GOOGLE, 2010). No entanto a API de sensores do Android define as suas coordenadas de forma relativa ao topo e à orientação do dispositivo (GOOGLE, 2010j). Quando a tela é ajustada pela API em resposta à mudança de orientação, o sistema de coordenadas do sensor já não fica mais de acordo com o sistema de coordenadas da tela, causando rotações não esperadas na tela de algumas aplicações (GOOGLE, 2010j). O exemplo de aplicativo descrito acima é demonstrado na Figura 6. Com a mudança de orientação o fio de prumo não está mais direcionando para o chão como deveria.



Fonte: Google (2010j).

Figura 6 - Sistema de coordenadas quando o dispositivo não está em sua orientação padrão

De acordo com Google (2010j), uma correção possível seria travar a tela em alguma orientação, não permitindo ao Android tratar essa mudança. Para isso altera-se o atributo `android:screenOrientation` no arquivo de XML chamado `AndroidManifest` ajustando para os valores `landscape` (fixando a tela na orientação *landscape*), `portrait` (fixando a tela

na orientação *portrait*) ou `user` (obtendo a orientação padrão escolhida pelo usuário).

As aplicações que não podem travar a tela em uma orientação fixa podem tentar compensar os valores do acelerômetro tanto para a orientação *landscape* quanto para a *portrait* (GOOGLE, 2010j). Segundo Google (2010j) a orientação real do dispositivo pode ser obtida através do método `getRotation()` e são quatro os valores possíveis de retorno desse método, `Surface.ROTATION_0` (quando não há rotação), `Surface.ROTATION_90`, `Surface.ROTATION_180`, ou `Surface.ROTATION_270`. Com o valor da rotação é possível compensar no eixo Z qualquer rotação que está sendo feita (GOOGLE, 2010j). Para evitar que o desenvolvedor de aplicativos faça cálculos matemáticos para essa compensação, está disponível o método `SensorManager.remapCoordinateSystem()` (GOOGLE, 2010j).

2.2.8 Desenvolvimento visando desempenho

Aplicações para dispositivos móveis com altas taxas de atualização de imagens gráficas, devem considerar a otimização no uso do processador, memória e bateria do dispositivo. Segundo Google (2010c), uma aplicação com boa otimização obedece aos princípios abaixo descritos.

O primeiro princípio é o de evitar a criação de objetos desnecessários (GOOGLE, 2010c). Esse princípio trata tanto da criação direta quanto da criação indireta de objetos. Um exemplo clássico de criação indireta se dá através de objetos da classe `String`. A classe `String` não admite mudança de estado, uma vez criado um objeto este sempre terá o mesmo valor (GUJ, 2010). Todos os métodos que efetuariam mudanças na `String`, não realizam isto sobre a `String` em questão, e sim sobre uma cópia dela, retornando esta cópia com a modificação pedida (GUJ, 2010). Para evitar a criação de objetos desnecessários a classe `StringBuffer` manipula cadeia de caracteres que, ao final de transformações pode ser convertido em `String`, evitando alocações desnecessárias (IMASTERS, 2010). Google (2010c) recomenda que, se um método tiver como retorno uma `String` e o resultado deste método sempre será adicionado a um `StringBuffer`, a mudança da assinatura do método para receber tal `StringBuffer` como parâmetro e a adição do objeto `String` no `StringBuffer` dentro do próprio método, é mais barata em termos de alocação de memória do que a criação da `String` temporária.

No que diz respeito à criação de vetores, os unidimensionais são muito mais eficientes

do que os multidimensionais (GOOGLE, 2010c). Nesses, o uso de tipos primitivos é recomendado ao invés do uso de classes, como é o exemplo do primitivo `int` e sua classe correspondente `Integer` (GOOGLE, 2010c).

O segundo princípio define que se um método não faz nenhum acesso aos atributos da classe, esse método deve ser estático, que no Java é representado pelo modificador `static` (GOOGLE, 2010c). Segundo Google (2010c), chamadas a métodos estáticos são de 15 a 20% mais rápidas do que chamadas a métodos não estáticos.

Nas linguagens orientadas a objetos é uma prática comum o uso de métodos `getters` e `setters`[8] ao invés de acessar os atributos diretamente (GOOGLE, 2010c). Essa é uma prática excelente, pois os compiladores normalmente conseguem otimizar o acesso e caso o programador quiser restringir ou depurar os acessos a determinado atributo, facilmente isso será possível através dos seus métodos acessores (GOOGLE, 2010c). Google (2010c) não recomenda essa prática no Android, pois as chamadas de métodos são caras, muito mais do que acessar atributos de classe diretamente. Os métodos acessores quando usados em interfaces públicas oferecem uma maneira mais sensata por permitir o uso do polimorfismo e herança, porém em uma classe a recomendação é de que os atributos sejam acessados diretamente (GOOGLE, 2010c). O acesso direto aos atributos de uma classe é quase sete vezes mais rápido do que o acesso através dos métodos `getters` e `setters` (GOOGLE, 2010c).

Ao declarar atributos que não terão mais seu valor alterado, a recomendação é utilizar os modificadores estático e final (GOOGLE, 2010c). Atributos estáticos que não são finais obrigam o compilador a gerar um inicializador para a classe, chamado `<clinit>` (GOOGLE, 2010c). Esse inicializador será executado no primeiro momento em que a classe é usada e tem a responsabilidade de atribuir os valores iniciais aos atributos estáticos (GOOGLE, 2010c). Tais atributos, quando referenciados, terão seu valor obtido através da busca pelo valor do atributo tanto na classe quanto nas super-classes, o que é desnecessário para um atributo cujo valor nunca será modificado (GOOGLE, 2010c). Quando utilizado o modificador final em atributos, o compilador substitui todas as chamadas desses atributos pelos seus respectivos valores (GOOGLE, 2010c). Essa otimização só é válida para atributos de tipos primitivos ou da classe `String`, porém é uma boa prática declarar atributos finais sempre que possível (GOOGLE, 2010c).

---

[8] Padrão de nomenclatura de métodos usado na orientação a objetos para encapsular os atributos de uma determinada classe. Comumente chamados de métodos acessores, possuem a responsabilidade de modificar um atributo, no caso do `setter`, ou disponibilizar o valor de um atributo, no caso do `getter` (TIEXPERT, 2010).

A partir da versão 1.5 do Java foi adicionada uma estrutura de repetição chamada de `enhanced for loop` cujo objetivo foi o de simplificar a sintaxe para percorrer vetores ou coleções que possuem iteradores. Segundo Google (2010c), o desempenho para esse tipo de *loop* é o mesma do que os outros tipos de *loops* da linguagem, salvo quando utilizada a classe `ArrayList` em que o uso de *loop* com contador é até três vezes mais rápido.

O uso de enumerações pode ajudar na legibilidade do código, mas uma aplicação que necessita de velocidade no processamento e possui pouco espaço, deve evitar esse recurso (GOOGLE, 2010c). Segundo Google (2010c), uma enumeração simples com três constantes pode adicionar até 740 *bytes* no arquivo compilado se for comparado com uma classe com três atributos públicos, estáticos e finais do tipo `int`.

É preciso tomar muito cuidado ao compartilhar atributos e métodos entre classes aninhadas (GOOGLE, 2010c). Quando a classe interna faz uso de algum atributo ou método da classe externa, a recomendação é de trocar a visibilidade do atributo para ser no mínimo `package` (GOOGLE, 2010c). O compilador considera que as duas classes são diferentes e quando a classe interna acessa atributos ou métodos privados da classe externa, este é obrigado a gerar métodos sintéticos que permitem tal acesso (GOOGLE, 2010c).

No que diz respeito ao uso de tipos numéricos, o uso de pontos flutuantes é pelo menos duas vezes mais lento do que o uso de variáveis do tipo inteiro (GOOGLE, 2010c). Alguns hardwares conseguem efetuar a multiplicação de números inteiros, porém a divisão e a operação de módulo ainda são efetuadas pelo software (GOOGLE, 2010c).

Google (2010c) recomenda ainda a preferência pelo uso do código disponível na biblioteca, favorecendo não só a reutilização de código como também da otimização que a biblioteca disponibiliza. Tal otimização não pode ser obtida com a implementação manual da mesma funcionalidade disponível na biblioteca. Um exemplo está no método `String.indexOf` no qual a máquina virtual Dalvik sobrescreve o código para otimização (GOOGLE, 2010c). Também o método `System.arraycopy` é quase nove vezes mais rápido do que se fosse escrito manualmente (GOOGLE, 2010c).

O uso de métodos nativos deve ser feito com cautela no Android, pois não necessariamente serão mais rápidos do que o código Java (GOOGLE, 2010c). Isso ocorre, pois a chamada JNI aos métodos nativos não permite ao compilador otimizar além das fronteiras do Java (GOOGLE, 2010c).

A última recomendação feita por Google (2010c) é a de tomar cautela e somente otimizar o código quando realmente for necessário, evitando que a aplicação fique difícil de ler e fazendo com que não seja possível medir os benefícios das otimizações descritas acima.

2.3 TRABALHOS CORRELATOS

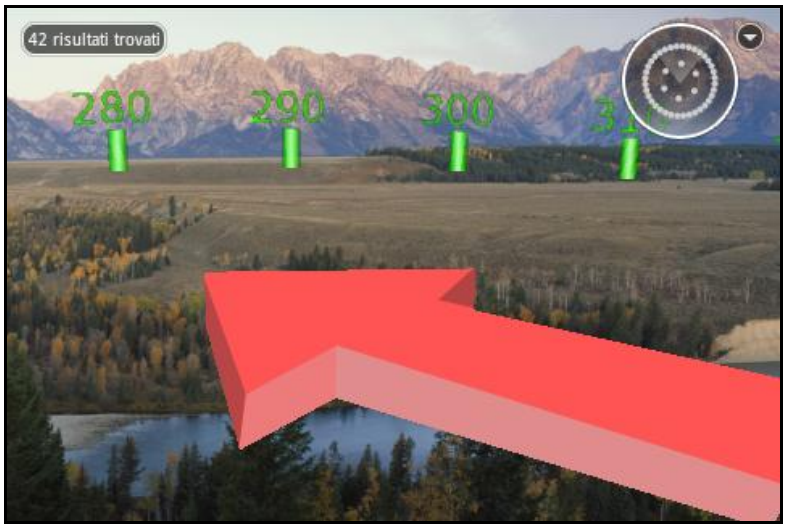
Existem vários projetos que fornecem suporte a criação de ambientes com realidade aumentada para computadores pessoais, tais como ARToolkit (ARTOOLKIT, 2010), ARTag (ARTAG, 2010) e Studierstube (TU GRAZ, 2010b). Especificamente para dispositivos móveis, pode-se citar: Studierstube ES (TU GRAZ, 2010a), Layar (LAYAR BV, 2010d) e Magnitude (MAGNITUDEHQ, 2010b).

Desenvolvido pela Graz University of Technology, Studerstube ES é um *framework* para desenvolvimento de aplicações com realidade aumentada para dispositivos móveis nas plataformas Windows CE, Symbian, iPhone, Linux e Windows 2k/XP. Ele oferece suporte à comunicação de rede para aplicações colaborativas, registro dos objetos virtuais através de marcações e desenvolvimento do aplicativo no próprio dispositivo móvel (TU GRAZ, 2010c).

Layar é um *framework* que permite o desenvolvimento de aplicativos com realidade aumentada através de um servidor próprio. Possui um software a ser instalado no dispositivo móvel que não necessita de customização por parte do desenvolvedor de aplicativo. Dentre os recursos disponíveis na plataforma pode-se citar (LAYAR BV, 2010a):

a) os objetos virtuais são chamados de pontos de interesse e estes são registrados através de coordenada geográfica no sítio Layar;
b) utiliza a câmera do dispositivo para capturar as imagens reais;
c) permite selecionar os objetos virtuais através do multitoque;
d) utiliza o GPS e o acelerômetro do dispositivo;
e) disponível nas plataformas iPhone 3GS e Android;
f) os pontos de interesse podem ser objetos 2D e 3D;
g) permite registrar um áudio a ser tocado quando aparece determinado ponto de interesse;
h) possui um serviço de autenticação para pontos de interesse privados;
i) permite visualizar os pontos de interesse através do serviço do Google Maps;
j) possui um serviço para testes do aplicativo desenvolvido.

Um dos maiores destaques do Layar é a possibilidade de renderização dos pontos de interesse em 3D, tornando a aplicação muito mais ampla e robusta. A Figura 7 mostra uma aplicação de uma bússola em 3D.



Fonte: Layar (2010c).

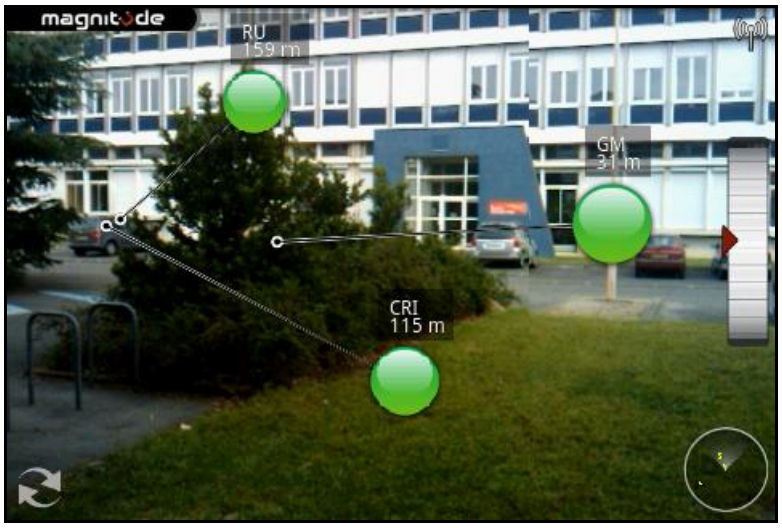
Figura 7 - Bússola em 3D desenvolvida no framework Layar

A Figura 8 mostra as antigas torres World Trade Center em 3D na localização exata onde estavam as torres originais, totalmente renderizadas no aplicativo Layar.

Fonte: Layar (2010b).

Figura 8 – Antigas torres do World Trace Center em 3D desenvolvidas no Layar

Magnitude é um projeto acadêmico *open-source* que visa fornecer serviços práticos de realidade aumentada pelos alunos do INSA Toulouse, criando uma estrutura modular e reutilizável para outras aplicações de realidade aumentada (MAGNITUDEHQ, 2010b). A Figura 9 mostra a tela principal da aplicação com os pontos de interesse destacados pelos círculos.



Fonte: Magnitudehq (2010a).

Figura 9 - Visualização de realidade aumentada pelo Magnitude

O framework Magnitude possui uma aplicação para dispositivos móveis da plataforma Android e também um servidor de aplicação Java *Enterprise Edition* (JEE) que informa os pontos de interesse mais adequados (MAGNITUDEHQ 2010c). A modularização foi um dos principais objetivos do projeto, tornando fácil a reutilização do aplicativo (MAGNITUDEHQ, 2010c). Os recursos da plataforma Android utilizados nessa aplicação foram: GPS, bússola, acelerômetro e câmera de vídeo (MAGNITUDE, 2010c).

3 DESENVOLVIMENTO

Neste capítulo são detalhadas as etapas do desenvolvimento do trabalho. São ilustrados os principais requisitos, a especificação, a implementação e por fim são listados resultados e discussão.

3.1 DESENVOLVIMENTO EM ANDROID

A documentação disponível na referência Google (2010e) explica de forma completa o download e a instalação do framework bem como a configuração do ambiente de desenvolvimento.

A documentação para depuração dos aplicativos em dispositivos, disponível na referência Google (2010n), não funcionou ao utilizar com um dispositivo HTC Desire na versão 2.2 do Androidem um notebook com sistema operacional Windows 7 *Home Premium* para 64 *bits*. Por outro lado o software PdaNet (2010) possui recursos para comunicação através da porta *Universal Serial Bus* (USB) com dispositivos da plataforma Android. As instruções para a instalação do PdaNet estão disponíveis na referência PdaNet (2010).

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O presente trabalho deverá:

a) permitir visualizar o ambiente real através da câmera do dispositivo (Requisito Funcional - RF);
b) sobrepor ao ambiente real, objetos virtuais em 2D (RF);
c) utilizar o multitoque para visualizar detalhes dos objetos virtuais (RF);
d) utilizar a estratégia de registro dos objetos virtuais através de coordenada geográfica (RF);
e) utilizar o acelerômetro para rastrear a direção que o usuário está visualizando com o dispositivo (Requisito Não Funcional - RNF);

f) obter a coordenada geográfica e informações dos objetos virtuais através da rede 3G e/ou *wireless* (RNF);
g) ser implementado usando a plataforma Android (RNF);
h) ser implementado usando o paradigma de intenções proposto pelo Android (RNF);
i) disponibilizar dentro do simulador os recursos de câmera, sensores e coordenadas geográficas (RF).

3.3 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida através da ferramenta StarUML, utilizando os conceitos de orientação a objetos e baseando-se nos diagramas da *Unified Modeling Language* (UML), gerando como produtos os diagramas de caso de uso, de classes e de seqüência apresentados nas seções seguintes.

3.3.1 Casos de uso

Nesta sessão são descritos os casos de uso de todos os recursos da aplicação. Foram identificados dois atores principais. O primeiro deles, o `Usuário`, faz uso dos recursos de realidade aumentada. Já o ator `Desenvolvedor` faz uso das ferramentas disponibilizadas para possibilitar o desenvolvimento de realidade aumentada no simulador do Android. Na Figura 10 é apresentado o diagrama de casos de uso da aplicação.

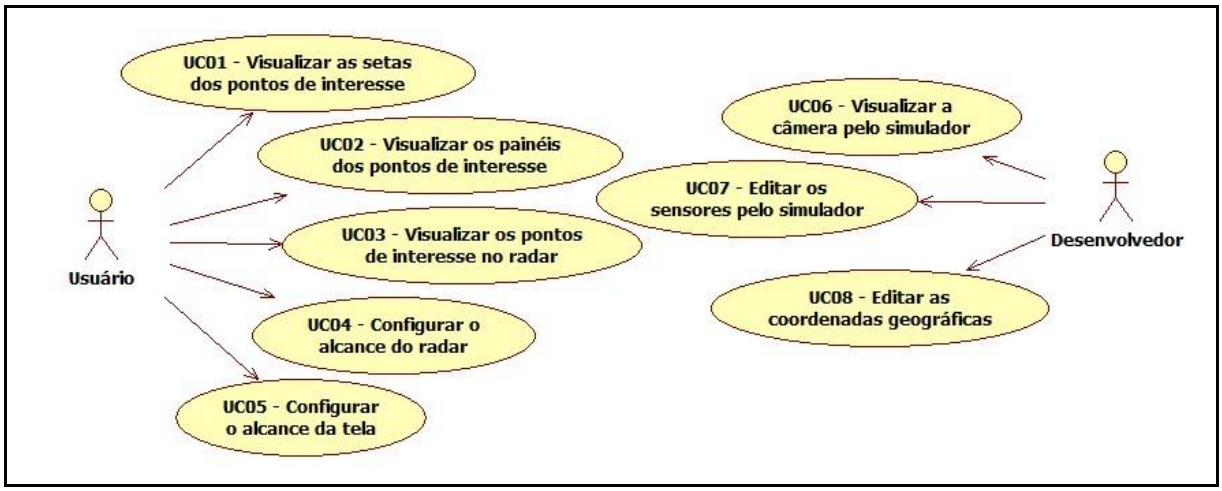


Figura 10 - Diagrama de casos de uso

3.3.1.1 Visualizar as setas dos pontos de interesse

Este caso de uso descreve como o ator `Usuário` interage com o aplicativo de realidade aumentada para visualizar as setas que apontam para a direção de cada ponto de interesse. Detalhes deste caso de uso estão descritos no Quadro 1.

UC01 – Visualizar as setas dos pontos de interesse	
Descrição	Uma das maneiras de visualizar os pontos de interesse é através das setas que partem do centro da tela e apontam para a direção dos respectivos pontos de interesse. Cada seta aponta para um ponto de interesse e possui a distância que este ponto se encontra do `Usuário`. Serão mostrados apenas os pontos de interesse que estiverem dentro do raio de alcance configurado no `UC05`.
Pré-Condição	O dispositivo deve possuir um serviço de localização geográfica, um sensor de acelerômetro e outro de orientação magnética, todos ativos.
Cenário Principal	1. O `Usuário` entra no aplicativo de realidade aumentada; 2. A aplicação exibe a tela inicial com um menu; 3. O `Usuário` pressiona o botão `Realidade Aumentada`; 4. A aplicação mostra a tela de realidade aumentada com imagens da câmera; 5. A aplicação carrega os pontos de interesse; 6. O `Usuário` aponta a câmera do celular para o chão; 7. A aplicação mostra as setas dos pontos de interesse encima das imagens da câmera. Cada seta deve possuir a distância que o ponto de interesse se encontra.
Cenário Alternativo 1	1. Se no passo 5 do cenário principal a aplicação não encontrar nenhum ponto de interesse dentro do alcance configurado, no passo 7 a aplicação não irá mostrar nenhuma seta; 2. Se no passo 4 o `Usuário` não apontar a câmera para o chão, as setas não irão aparecer na tela.
Cenário Alternativo 2	1. Se o `Usuário` já estiver com o aplicativo de realidade aumentada aberto, pular para o passo 6.
Pós-Condição	O `Usuário` deve poder enxergar as setas apontando para a direção dos pontos de interesse que se encontram dentro do raio de alcance configurado.

Quadro 1 - Caso de uso UC01

3.3.1.2 Visualizar os painéis dos pontos de interesse

Este caso de uso descreve como o ator `Usuário` interage com o aplicativo de realidade aumentada para visualizar os painéis de cada ponto de interesse. Detalhes deste caso de uso estão descritos no Quadro 2.

UC02 – Visualizar os painéis dos pontos de interesse	
Descrição	Uma outra maneira de visualizar os pontos de interesse é através dos painéis pintados na direção de cada ponto de interesse. Cada painel possui o nome do ponto de interesse respectivo. Serão mostrados apenas os pontos de interesse que estiverem dentro do raio de alcance configurado no `UC05`.
Pré-Condição	O dispositivo deve possuir um serviço de localização geográfica, um sensor de acelerômetro e outro de orientação magnética, todos ativos.
Cenário Principal	1. O `Usuário` entra no aplicativo de realidade aumentada; 2. A aplicação exibe a tela inicial com um menu; 3. O `Usuário` pressiona o botão `Realidade Aumentada`; 4. A aplicação mostra as a tela de realidade aumentada com imagens da câmera; 5. A aplicação carrega os pontos de interesse; 6. O `Usuário` aponta a câmera do celular para frente, na altura dos ombros; 7. A aplicação mostra os painéis dos pontos de interesse encima das imagens da câmera. Cada painel deve possuir o nome do ponto de interesse respectivo.
Cenário Alternativo 1	1. Se no passo 5 do cenário principal a aplicação não encontrar nenhum ponto de interesse, no passo 7 a aplicação não irá mostrar nenhum painel; 2. Se no passo 6 o `Usuário` não apontar a câmera para frente, os painéis não irão aparecer na tela.
Cenário Alternativo 2	1. Se o `Usuário` já estiver com o aplicativo de realidade aumentada aberto, pular para o passo 6.
Pós-Condição	O `Usuário` deve poder enxergar os painéis dos pontos de interesse que estão na direção da câmera e que se encontram dentro do raio de alcance configurado.

Quadro 2 - Caso de uso UC02

3.3.1.3 Visualizar os pontos de interesse no radar

Este caso de uso descreve como o ator `Usuário` interage com o aplicativo de realidade aumentada para visualizar os pontos de interesse no radar. Detalhes deste caso de uso estão descritos no Quadro 3.

UC03 – Visualizar os pontos de interesse no radar	
Descrição	A última maneira de visualizar os pontos de interesse é através de pequenos pontos pintados no radar. Serão mostrados apenas os pontos de interesse que estiverem dentro do raio de alcance do radar, configurado no `UC04`. O radar mostra os quatro pontos cardeais principais (norte, sul, leste e oeste) e os pontos desenhados obedecem à orientação magnética.
Pré-Condição	O dispositivo deve possuir um serviço de localização geográfica, um sensor de acelerômetro e outro de orientação magnética, todos ativos.

Cenário Principal	1. O `Usuário` entra no aplicativo de realidade aumentada; 2. A aplicação exibe a tela inicial com um menu; 3. O `Usuário` pressiona o botão `Realidade Aumentada`; 4. A aplicação mostra as imagens da câmera e no canto direito superior o radar; 5. A aplicação carrega os pontos de interesse no radar.
Cenário Alternativo 1	1. Se no passo 5 do cenário principal a aplicação não encontrar nenhum ponto de interesse, nenhum ponto será mostrado no radar.
Cenário Alternativo 2	1. Se após o passo 5 do cenário principal o `Usuário` movimentar-se, alterando a orientação magnética, os pontos pintados no radar devem movimentar-se igualmente, obedecendo a direção real dos pontos de interesse.
Pós-Condição	O `Usuário` deve poder enxergar o desenho de um ponto no radar para cada ponto de interesse cuja distância estiver dentro do alcance do radar.

Quadro 3 - Caso de uso UC03

3.3.1.4 Configurar o alcance do radar

Este caso de uso descreve como o ator `Usuário` configura o raio de alcance do radar. Detalhes deste caso de uso estão descritos no Quadro 4.

UC04 – Configurar o alcance do radar	
Descrição	O raio de alcance do radar é um valor medido em metros que delimita quantos e quais pontos de interesse serão pintados no radar.
Pré-Condição	O aplicativo de realidade aumentada deve estar aberto em sua tela inicial.
Cenário Principal	1. O `Usuário` toca no botão `Configurações`; 2. A aplicação exibe a tela de configurações; 3. O `Usuário` toca na configuração `Alcance do Radar`; 4. A aplicação exibe uma tela modal com um teclado virtual, mostrando o valor atual do alcance do radar a ser configurado; 5. O `Usuário` toca nos números do teclado para informar o novo alcance do radar; 6. O `Usuário` toca no botão `OK` para confirmar; 7. A aplicação atualiza o radar com os pontos;

Cenário Alternativo 1	Uma segunda maneira de configurar o alcance do radar é na própria tela de realidade aumentada, conforme descrito neste cenário. 1. O `Usuário` toca no botão `Realidade Aumentada`; 2. A aplicação abre a tela de realidade aumentada com o desenho do radar no canto direito acima; 3. O `Usuário` toca no desenho do radar; 4. A aplicação exibe uma tela modal mostrando o raio de alcance atual; 5. O `Usuário` toca no raio de alcance atual; 6. A aplicação exibe um teclado virtual para informar o novo alcance; 7. O `Usuário` toca nos números do teclado para informar o novo alcance; 8. O `Usuário` toca no botão que possui o desenho de um teclado, para confirmar; 9. A aplicação fecha o teclado virtual e exibe a tela modal com o novo raio de alcance; 10. O `Usuário` toca no botão `Salvar` para confirmar; 11. A aplicação atualiza o radar com os pontos.
Cenário Alternativo 2	1. Se nos passos 4 ou 6 do cenário principal ou nos passos 5 ou 10 do cenário alternativo 1 o `Usuário` tocar no botão `Cancelar`, o valor do alcance do radar permanecerá sem alteração.
Cenário Alternativo 3	1. Se no passo 4 do cenário principal ou no passo 3 do cenário alternativo 1 o `Usuário` tocar no botão `Salvar`, o valor do alcance do radar permanecerá sem alteração.
Cenário Alternativo 4	1. Se no passo 5 do cenário principal ou no passo 7 do cenário alternativo 1 o `Usuário` tocar em botões de letras ou símbolos, o teclado virtual não irá exibi-los pois apenas valores numéricos são aceitos.
Pós-Condição	O radar deve estar atualizado, mostrando apenas os pontos de interesse que atendem ao novo raio de alcance.

Quadro 4 - Caso de uso UC04

3.3.1.5 Configurar o alcance da tela

Este caso de uso descreve como o ator `Usuário` configura o alcance da tela. Detalhes deste caso de uso estão descritos no Quadro 5.

UC05 – Configurar o alcance da tela	
Descrição	O alcance da tela é um valor medido em metros que delimita até que distância os pontos de interesse serão carregados, de forma a garantir que não serão carregados pontos de interesse muito distantes da localização do dispositivo.
Pré-Condição	O aplicativo de realidade aumentada deve estar aberto em sua tela inicial.

Cenário Principal	1. O `Usuário` toca no botão `Configurações`; 2. A aplicação exibe a tela de configurações; 3. O `Usuário` toca na configuração `Alcance da Tela`; 4. A aplicação exibe uma tela modal com um teclado virtual, mostrando o valor atual do alcance da tela a ser configurado; 5. O `Usuário` toca nos números do teclado para informar o novo alcance da tela; 6. O `Usuário` toca no botão `OK` para confirmar; 7. A aplicação atualiza o radar com os pontos;
Cenário Alternativo 1	Uma segunda maneira de configurar o alcance da tela é na própria tela de realidade aumentada, conforme descrito neste cenário. 1. O `Usuário` toca no botão `Realidade Aumentada`; 2. A aplicação abre a tela de realidade aumentada com o texto `Alcance` no canto esquerdo abaixo; 3. O `Usuário` toca no texto `Alcance`; 4. A aplicação exibe uma tela modal mostrando o alcance atual; 5. O `Usuário` toca no alcance atual; 6. A aplicação exibe um teclado virtual para informar o novo alcance; 7. O `Usuário` toca nos números do teclado para informar o novo alcance; 8. O `Usuário` toca no botão que possui o desenho de um teclado, para confirmar; 9. A aplicação fecha o teclado virtual e exibe a tela modal com o novo alcance; 10. O `Usuário` toca no botão `Salvar` para confirmar; 11. A aplicação atualiza o alcance.
Cenário Alternativo 2	1. Se nos passos 4 ou 6 do cenário principal ou nos passos 5 ou 10 do cenário alternativo 1 o `Usuário` tocar no botão `Cancelar`, o valor do alcance permanecerá sem alteração.
Cenário Alternativo 3	1. Se no passo 4 do cenário principal ou do cenário alternativo o `Usuário` tocar no botão `Salvar`, o valor do alcance permanecerá sem alteração.
Cenário Alternativo 4	1. Se no passo 5 do cenário principal ou no passo 7 do cenário alternativo 1 o `Usuário` tocar em botões de letras ou símbolos, o teclado virtual não irá exibi-los pois apenas valores numéricos são aceitos.
Pós-Condição	A tela deve estar atualizada, mostrando apenas os pontos de interesse cuja distância é menor do que o valor do alcance configurado.

Quadro 5 - Caso de uso UC05

3.3.1.6 Visualizar a câmera pelo simulador

Este caso de uso descreve como o ator `Desenvolvedor` utiliza uma câmera para rodar o aplicativo de realidade aumentada dentro do simulador do Android. Detalhes deste caso de uso estão descritos no Quadro 6.

UC06 – Visualizar a câmera pelo simulador	
Descrição	A necessidade deste caso de uso se dá pela falta de suporte aos dispositivos de câmera por parte do simulador do Android. Neste caso de uso, o aplicativo de realidade aumentada será utilizado exclusivamente dentro do simulador do Android.
Pré-Condição	1. O computador do `Desenvolvedor` deve possuir uma câmera, podendo esta ser interna ou externa; 2. O computador do `Desenvolvedor` deve estar conectado em alguma *Local Area Network* (LAN) e nela deve possuir um IP; 3. Deverá ser utilizado o simulador do Android.
Cenário Principal	1. O `Desenvolvedor` abre o aplicativo de câmera disponibilizado neste trabalho; 2. O aplicativo de câmera abre uma tela, pedido para informar um dispositivo de vídeo; 3. O `Desenvolvedor` seleciona o dispositivo de vídeo mais adequado e pressiona o botão `OK`; 4. O aplicativo de câmera começa a disponibilizar as imagens da câmera na porta 9889; 5. O `Desenvolvedor` entra no aplicativo de realidade aumentada dentro do simulador do Android; 6. A aplicação de realidade aumentada exibe a tela inicial com um menu; 7. O `Desenvolvedor` toca o botão `Realidade Aumentada`; 8. A aplicação de realidade aumentada busca as imagens da câmera no computador e na porta 9889.
Cenário Alternativo 1	Para configurar o IP e a porta que serão usadas na conexão que obtém as imagens da câmera, no passo 7 do cenário principal devem ser executados os passos descritos neste cenário. 1. O `Desenvolvedor` toca no botão `Configurações`; 2. A aplicação de realidade aumentada exibe a tela de configurações; 3. O `Desenvolvedor` toca na configuração `IP da fonte de câmera` ou `Porta da fonte de câmera`; 4. A aplicação de realidade aumentada exibe uma tela modal para configuração do IP ou da porta; 5. O `Desenvolvedor` toca nos números do teclado para configurar o novo IP ou a nova porta; 6. O `Desenvolvedor` toca no desenho de um teclado para confirmar; 7. A aplicação de realidade aumentada atualiza a configuração do IP ou da porta; 8. O `Desenvolvedor` pressiona o botão `return` para voltar na tela de menu. Voltar para o passo 7 do cenário principal.
Exceção 1	1. Se no passo 3 do cenário principal o `Desenvolvedor` pressionar o botão `Cancelar`, a aplicação de câmera lançará uma mensagem na tela e terá sua execução abortada.
Exceção 2	1. Se no passo 4 a porta 9889 já estiver sendo usada ou não estiver disponível, a aplicação de câmera lançará uma mensagem na tela e terá sua execução abortada.

Exceção 3	1. Se no passo 8 do cenário principal a aplicação de realidade aumentada não encontrar as imagens da câmera através do IP e da porta 9889, será lançada uma mensagem na tela e sua execução será abortada.
Pós-Condição	1. A aplicação de câmera deve disponibilizar as imagens da câmera pelo IP do computador e pela porta 9889; 2. A aplicação de realidade aumentada deve receber as imagens da câmera disponibilizadas pela aplicação de câmera.

Quadro 6 - Caso de uso UC06

3.3.1.7 Editar os sensores pelo simulador

Este caso de uso descreve como o ator `Desenvolvedor` utiliza uma aplicação externa para simular os sensores quando rodar o aplicativo de realidade aumentada dentro do simulador do Android. Detalhes deste caso de uso estão descritos no Quadro 7.

UC07 – Editar os sensores pelo simulador	
Descrição	A necessidade deste caso de uso se dá pela falta de suporte à simulação dos sensores por parte do simulador do Android. Neste caso de uso, o aplicativo de realidade aumentada será utilizado exclusivamente dentro do simulador do Android.
Pré-Condição	1. O computador do `Desenvolvedor` deve estar conectado em alguma LAN e nela deve possuir um IP; 2. Deverá ser utilizado o simulador do Android.
Cenário Principal	1. O `Desenvolvedor` abre o aplicativo de sensores disponibilizado neste trabalho; 2. O aplicativo de sensores abre sua tela principal; 3. O `Desenvolvedor` marca os sensores que deseja habilitar em `Supported sensors` e em `Enabled sensors`; 4. O aplicativo de sensores começa a disponibilizar os valores dos sensores na porta 8010; 5. O `Desenvolvedor` entra no aplicativo de realidade aumentada dentro do simulador do Android; 6. A aplicação exibe a tela inicial com um menu; 7. O `Desenvolvedor` toca no botão `Realidade Aumentada`; 8. A aplicação de realidade aumentada busca os valores dos sensores no IP do computador e na porta 8010. 9. O `Desenvolvedor` interage com a aplicação de sensores, alterando os valores de *yaw*, *pitch* e *roll*. 10. A aplicação de realidade aumentada trata os valores de *yaw*, *pitch* e *roll* recebidos da aplicação de sensores.

Cenário Alternativo 1	Para configurar o IP e a porta que serão usadas na conexão que obtém os valores dos sensores, no passo 7 do cenário principal devem ser executados os passos descritos neste cenário. 1. O `Desenvolvedor` toca no botão `Configurações`; 2. A aplicação de realidade aumentada exibe a tela de configurações; 3. O `Desenvolvedor` toca na configuração `IP da fonte de sensores` ou `Porta da fonte de sensores`; 4. A aplicação de realidade aumentada exibe uma tela modal para configuração do IP ou da porta; 5. O `Desenvolvedor` toca nos números do teclado para configurar o novo IP ou a nova porta; 6. O `Desenvolvedor` toca no desenho de um teclado para confirmar; 7. A aplicação de realidade aumentada atualiza a configuração do IP ou da porta; 8. O `Desenvolvedor` pressiona o botão `return` para voltar na tela de menu. Voltar para o passo 7 do cenário principal.
Cenário Alternativo 2	1. O passo 3 é opcional, por padrão a aplicação já vem com todos os sensores habilitados. Pular para o passo 4.
Cenário Alternativo 3	1. Se no passo 4 a porta 8010 já estiver sendo usada ou não estiver disponível, a aplicação de sensores lançará a mensagem `Could not listen on port: 8010` na tela.
Exceção	2. Se no passo 8 do cenário principal a aplicação de realidade aumentada não encontrar os valores dos sensores através do IP e da porta 8010, será lançada uma mensagem na tela e sua execução será abortada.
Pós-Condição	1. A aplicação de sensores deve estar disponibilizando os valores dos sensores pelo IP do computador e pela porta 8010; 2. A aplicação de realidade aumentada deve estar recebendo os valores dos sensores disponibilizados pela aplicação de sensores.

Quadro 7 - Caso de uso UC07

3.3.1.8 Editar as coordenadas geográficas

Este caso de uso descreve como o ator `Desenvolvedor` interage com o aplicativo de realidade aumentada, simulando as coordenadas geográficas. Detalhes deste caso de uso estão descritos no Quadro 8.

UC08 – Editar as coordenadas geográficas	
Descrição	Através deste caso de uso, é possível simular os valores das coordenadas geográficas dentro da aplicação de realidade aumentada. A necessidade de simulação das coordenadas geográficas se dá quando a aplicação de realidade aumentada é utilizada no exclusivamente simulador do Android.
Pré-Condição	Deverá ser utilizado o simulador do Android.

Cenário Principal	1. O `Desenvolvedor` entra no aplicativo de realidade aumentada dentro do simulador do Android; 2. A aplicação exibe a tela inicial com um menu; 3. O `Desenvolvedor` toca no botão `Configurações`; 4. A aplicação exibe a tela de configurações; 5. O `Desenvolvedor` toca nos botões `Latitude` ou `Longitude`; 6. A aplicação exibe uma tela modal para simulação da latitude ou da longitude; 7. O `Desenvolvedor` toca nos números do teclado para configurar os valores de latitude ou longitude; 8. O `Desenvolvedor` toca no desenho de um teclado para confirmar; 9. A aplicação atualiza a configuração da latitude ou da longitude; 10. O `Desenvolvedor` pressiona o botão `return` para voltar à tela de menu; 11. A aplicação exibe a tela inicial com um menu; 12. O `Desenvolvedor` toca no botão `Realidade Aumentada`; 13. A aplicação exibe a tela de realidade aumentada; 14. O `Desenvolvedor` pressiona no simulador o botão `DPad left`; 15. A aplicação diminui uma medida de longitude e atualiza os pontos de interesse da tela; 16. O `Desenvolvedor` pressiona no simulador o botão `DPad right`; 17. A aplicação aumenta uma medida de longitude e atualiza os pontos de interesse da tela; 18. O `Desenvolvedor` pressiona no simulador o botão `DPad down`; 19. A aplicação diminui uma medida de latitude e atualiza os pontos de interesse da tela; 20. O `Desenvolvedor` pressiona no simulador o botão `DPad up`; 21. A aplicação aumenta uma medida de latitude e atualiza os pontos de interesse da tela.
Pós-Condição	A aplicação deve atualizar a localização do dispositivo de acordo com a coordenada geográfica configurada.

Quadro 8 - Caso de uso UC08

3.3.2 Diagramas de classes

Nesta sessão são descritas as classes necessárias para o desenvolvimento do aplicativo de realidade aumentada deste trabalho, o relacionamento entre elas e as suas estruturas. Para facilitar o entendimento de como as classes estão reunidas, na Figura 11 estão sendo demonstrados os pacotes, suas dependências bem como as classes que os compõem. O programa servidor que fornece pontos de interesse através da internet não está sendo representado na Figura 11 por não possuir relacionamento direto com as classes e pacotes do aplicativo de realidade aumentada, por esse motivo sua descrição está sendo feita no final desta sessão.

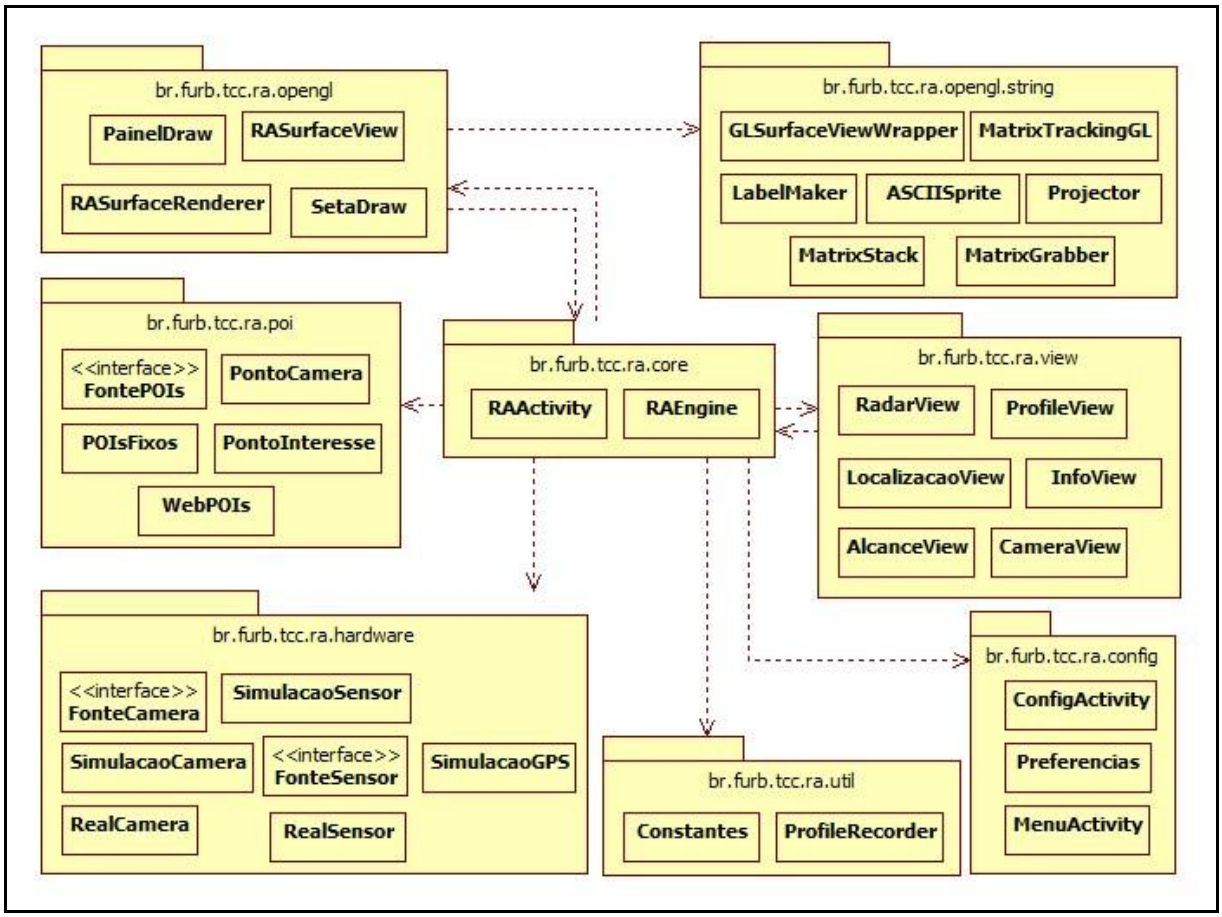


Figura 11 - Diagrama de pacotes do aplicativo de realidade aumentada

3.3.2.1 Pacote `br.furb.ra.core`

O primeiro pacote (Figura 12) é denominado `br.furb.ra.core` e possui as duas principais classes deste trabalho.

A classe `RAActivity` possui a responsabilidade de manter o ciclo de vida do programa bem como criar e interligar os objetos que dão vida a aplicação. Esta classe é uma extensão da classe `Activity` do Android e através dos métodos `onCreate`, `onDestroy`, `onPause`, `onResume` e `onStop` faz as devidas tratativas do ciclo de vida, como pausar e iniciar a renderização da câmera e da OpenGL bem como desligar e reiniciar a execução da classe `RAEngine`. Em especial o método `onCreate` configura o comportamento da atividade Android, definindo a orientação padrão da tela, colocando a aplicação em tela cheia e ajustando o *layout* da tela.

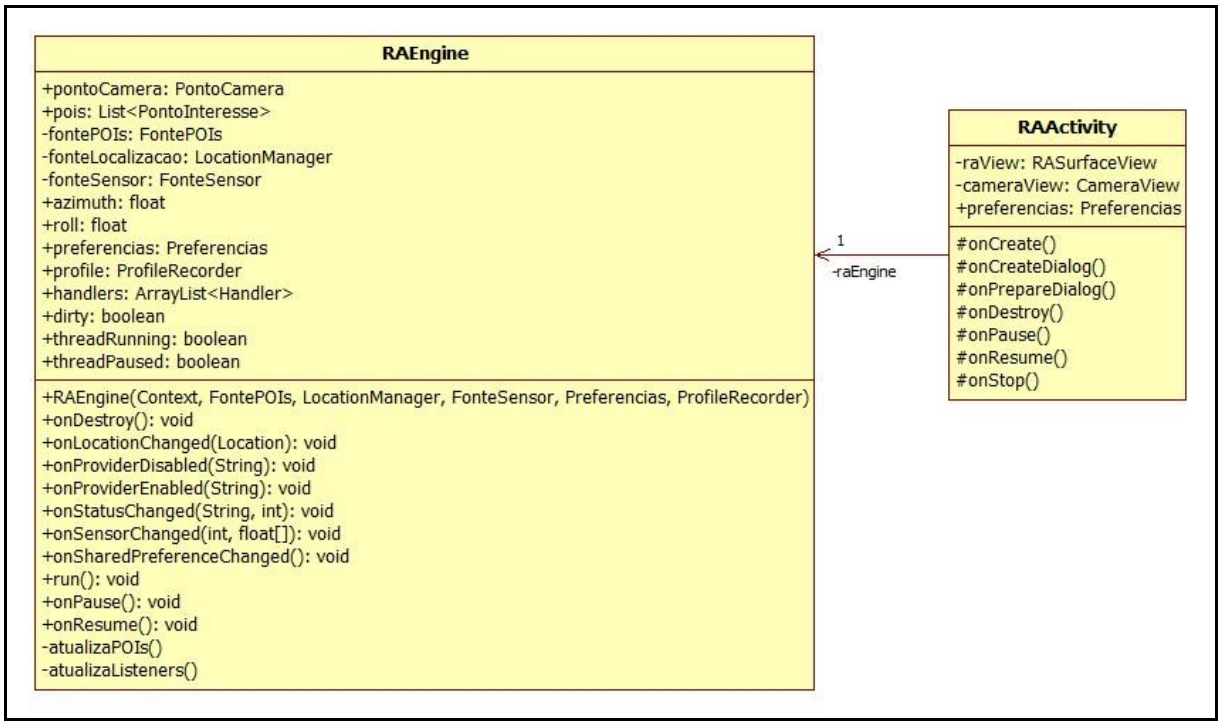


Figura 12 - Pacote `br.furb.ra.core`

O *layout* da tela, representado pela Figura 13 possui três camadas sobrepostas de forma a causar a impressão de aumento de realidade.

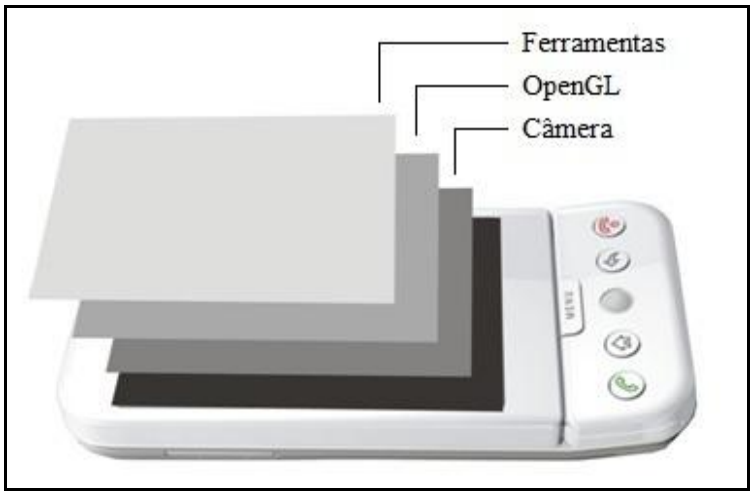


Figura 13 - Camadas da tela

A camada mais interior é responsável por desenhar as imagens recebidas pela câmera, a camada intermediária utiliza a OpenGL com um plano tridimensional que possui os pontos de interesse e a terceira camada possui o desenho das ferramentas de configuração como por exemplo o radar.

Por outro lado, a classe `RAEngine` possui a responsabilidade de executar todos os cálculos da aplicação, preparando o posicionamento de cada objeto que será mostrado na tela. Essa classe obtém todas as informações de mudanças nos sensores, coordenadas geográficas e das preferências, fazendo tratativas nas informações recebidas e divulgando para as demais

classes da aplicação. É através dos métodos `onSharedPreferenceChanged`, `onLocationChanged`, `onProviderDisabled`, `onProviderEnabled` e `onStatusChanged` que o Android avisa a classe `RAEngine` das mudanças. Os métodos `onDestroy` e `onPause` são chamados pela classe `RAActivity` para que essa classe possa desregistrar-se como observador de mudanças do dispositivo Android. O método `atualizaPOIs` é chamado pelos demais métodos quando houver alguma mudança significativa no dispositivo de forma que necessite de um novo cálculo para a localização e direção de cada ponto de interesse na tela. Por último o método `atualizaListeners` informa a todos os observadores desta classe que houve alguma mudança significativa nos pontos de interesse.

Os cálculos da *engine* são executados através de uma *thread* que possui o ciclo de vida representado pelo diagrama de estados da Figura 14.

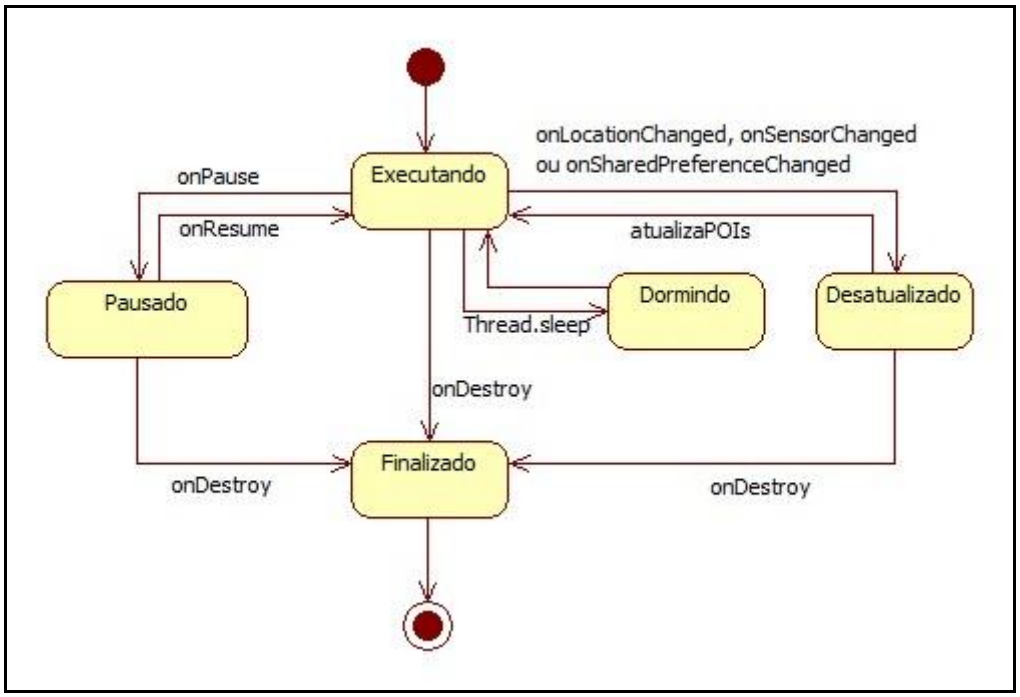


Figura 14 - Diagrama de estados da *engine*

O primeiro estado chama-se `Executando` e determina que a *engine* está em execução e aguardando por mudanças no dispositivo. Toda mudança que ocorrer no dispositivo e que a *engine* está aguardando, seja ela do acelerômetro, da bússola, da localização geográfica ou das preferências, faz a *engine* passar para o estado `Desatualizado`. Esse estado determina se os pontos de interesse estão desatualizados e necessitam de um novo cálculo para serem mostrados na tela. Para sair do estado `Desatualizado` e voltar ao estado `Executando` a *engine* executa o método `atualizaPOIs`, que calcula a nova posição de cada ponto de interesse na tela. Para não consumir de forma excessiva o processador do dispositivo, a *engine* entra no estado `Dormindo` através do método `Thread.sleep` e fica oscilando entre os estados

`Dormindo` e `Executando`. O estado `Pausado` é utilizado quando o método `onPause` é chamado pela classe `RAActivity` e indica que o usuário decidiu sair da aplicação sem finalizá-la. Nesse estado a aplicação ignora as mudanças no dispositivo. Quando o usuário retorna para a aplicação o método `onResume` é chamado pela classe `RAActivity` e o estado passa novamente a ser `Executando`. Quando o usuário decidir finalizar a aplicação o método `onDestroy` é chamado pela classe `RAActivity` fazendo com que o estado fique `Finalizado` e a execução da *engine* seja terminada.

3.3.2.2 Pacote `br.furb.ra.opengl`

As classes do pacote `br.furb.ra.opengl` são responsáveis por renderizar a camada de OpenGL da tela e estão representadas na Figura 15.

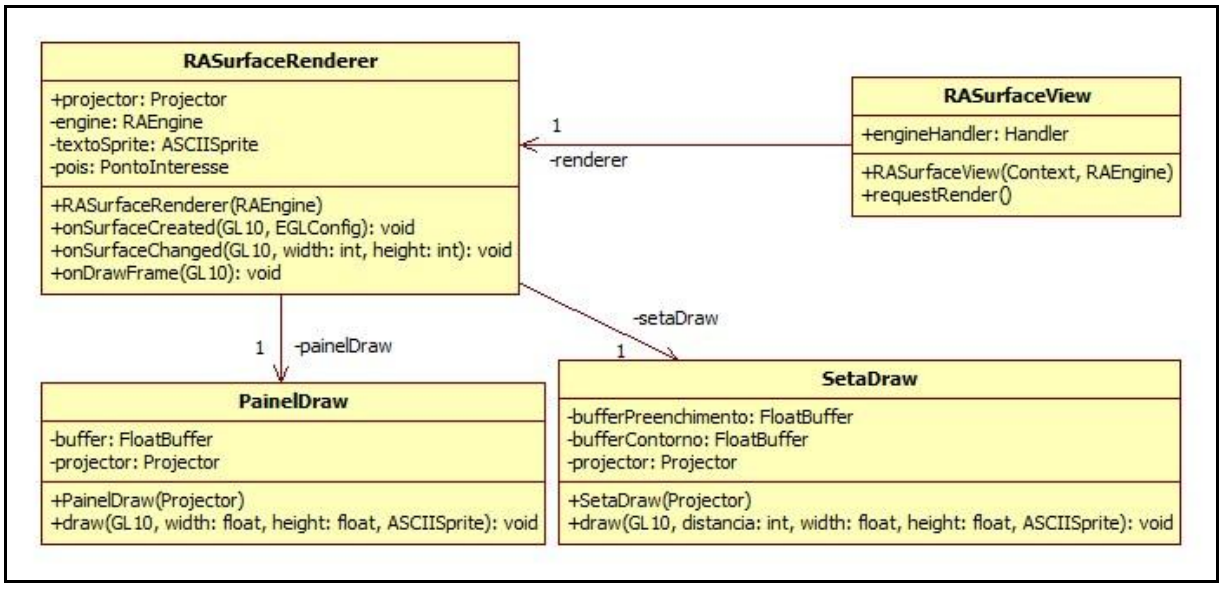


Figura 15 - Pacote `br.furb.ra.opengl`

A classe `RASurfaceView` é instanciada na `RAActivity` e a partir dela são instanciadas as demais. Essa classe é uma extensão da classe do Android chamada `GLSurfaceView` que trata de todo o ciclo de vida da OpenGL. Nela a OpenGL é configurada para ser translúcida e somente renderizar os objetos quando o método `requestRenderer` for chamado. Através do atributo `engineHandler` essa classe é avisada que houve alguma mudança na *engine* e portanto a OpenGL deve ser atualizada através da chamada ao método `requestRenderer`.

A classe `RASurfaceRenderer` tem por objetivo renderizar as setas e os painéis dos pontos de interesse na OpenGL e seus métodos são uma implementação da interface `GLSurfaceRenderer`. O método `onSurfaceCreated` prepara a OpenGL configurando, entre

outras coisas, a pintura de textura 2D e o `depth test`. Já o método `onSurfaceChanged` recebe as dimensões da tela, prepara a `viewport`, carrega a matriz de projeção, carrega a matriz dos modelos e prepara o `frustum`. Por último o método `onDrawFrame` é chamado apenas quando é requisitada a renderização sendo sua principal responsabilidade realizar todas as transformações e desenhar os painéis e as setas de cada ponto de interesse no seu devido lugar da tela.

A classe `SetaDraw` possui a responsabilidade de pintar uma seta com o texto da distância que o ponto de interesse está do dispositivo. A seta é desenhada com vetor de vértice, este sendo preparado no construtor da classe e atribuído aos atributos `buffer` e `bufferContorno` sendo que o primeiro desenha o preenchimento da seta através de triângulos e o segundo desenha o contorno da seta através de linhas. O texto da distância é desenhado através dos recursos do pacote `br.furb.ra.opengl.string` sendo utilizada, principalmente, a classe `ASCIISprite`.

Por último, a classe `PainelDraw` é responsável por desenhar um painel com o texto do nome do ponto de interesse representado. O painel também é desenhado com vetor de vértice preparado no construtor da classe, sendo armazenado no atributo `buffer` e desenhado através de dois triângulos. O texto do nome do ponto de interesse é desenhado também através da classe `ASCIISprite`.

3.3.2.3 Pacote `br.furb.ra.opengl.string`

Devido a limitação da especificação da OpenGL ES 1.0 em não possuir recursos para desenhar texto, foi necessário pesquisar as estratégias existentes para atender a essa necessidade. Uma das estratégias encontradas está disponível na referência Google (2010m), encontrada na API de *samples*. Foi necessário adaptar a implementação para a necessidade de desenhar textos de forma dinâmica, uma vez que os pontos de interesse poderiam variar na medida em que o dispositivo se move. A Figura 16 mostra o relacionamento entre as classes deste pacote.

Esse pacote possui três conjuntos de classes para atender a funcionalidade de desenho do texto na OpenGL. O primeiro conjunto encapsula o objeto de `GL10` e guarda as transformações ocorridas nele para que se possa transformar o texto da mesma forma. O segundo conjunto é responsável por avisar o primeiro conjunto de que houve uma série de

transformações relevantes e que as matrizes internas devem ser atualizadas. O terceiro conjunto tem por responsabilidade desenhar os textos através de texturas e replicar as transformações ocorridas na OpenGL para esses textos.

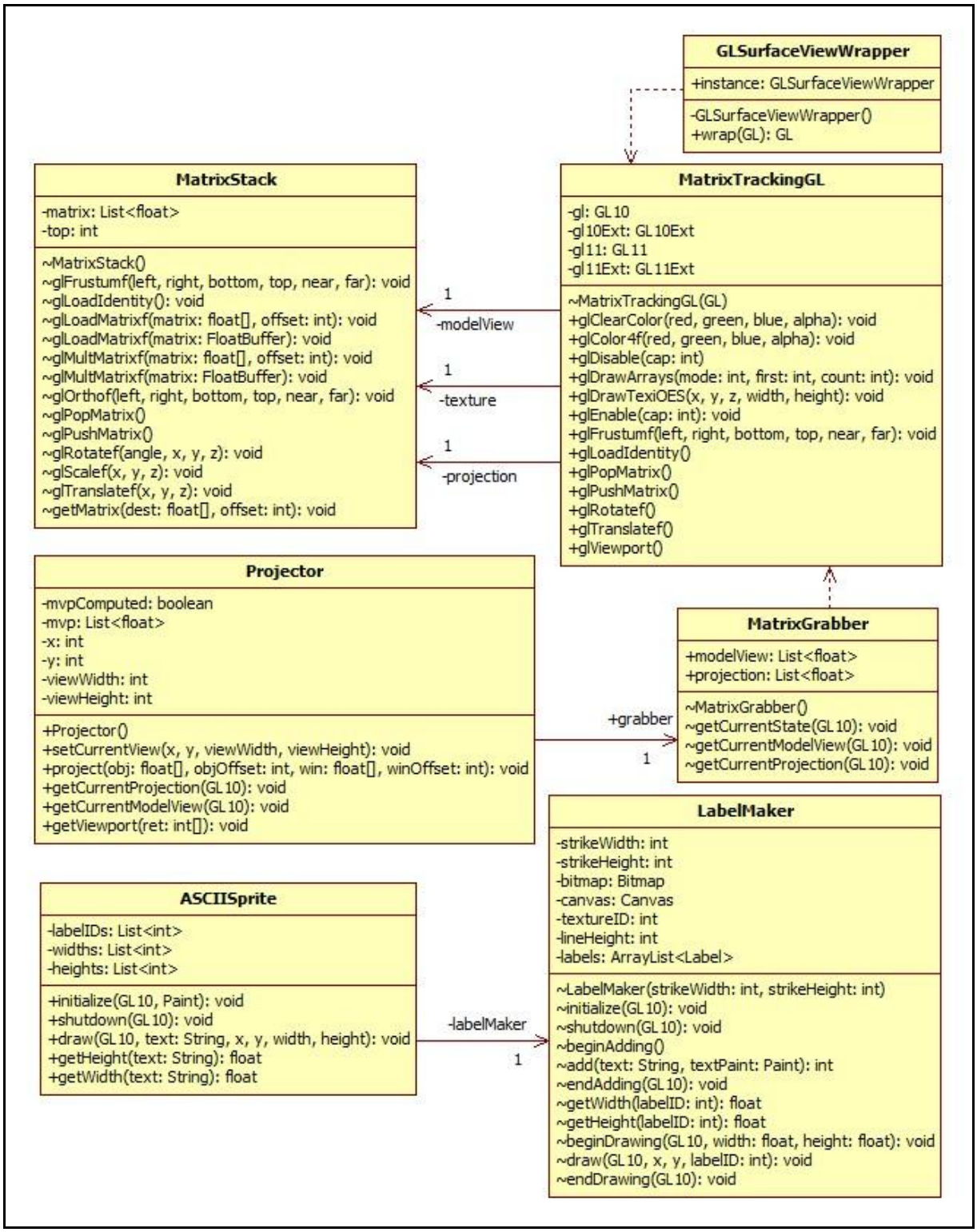


Figura 16 - Pacote `br.furb.ra.opengl.string`

No primeiro conjunto, a classe `GLSurfaceViewWrapper` é responsável por receber um objeto de `GL10` e encapsulá-lo em um objeto da classe `MatrixTrackingGL` através do padrão

de projeto *Wrapper*. Essa classe é uma implementação da interface `GLWrapper` que define o método `wrap` e sua implementação realiza toda a responsabilidade da classe. Como não é necessário instanciar essa classe mais de uma vez, ela foi implementada para ser *Singleton*.

A classe `MatrixTrackingGL` ao ser instanciada recebe o objeto `GL10` que será utilizado como objeto principal em todos os métodos dessa classe, utilizando o padrão de projeto *Delegate*. A responsabilidade dessa classe é direcionar as chamadas de OpenGL para o objeto `GL10` e atualizar suas matrizes internas com o objetivo de guardar as transformações ocorridas para que quando for requisitado, seja possível realizar tais transformações nos textos a serem desenhados na tela. As transformações são guardadas em três matrizes, objetos da classe `MatrixStack`, armazenadas nos atributos `modelView`, `texture` e `projection`.

A classe `MatrixStack` possui uma matriz para armazenar todas as transformações que são recebidas através de seus métodos. Sua responsabilidade é manter atualizada essa matriz de acordo com as transformações do objeto `GL10` recebidas através da classe `MatrixTrackingGL` nos seus métodos.

No segundo conjunto, a classe `Projector` tem a responsabilidade de receber a informação de que houveram transformações da OpenGL e atualizar as classes do primeiro conjunto. Quando houver alguma mudança na projeção do `GL10` deve ser chamado o método `getCurrentProjection`. Sempre que houver uma mudança no modelo do `GL10` deve ser chamado o método `getCurrentModelView` e quando houver alguma mudança na `viewport` do `GL10` deve ser chamado o método `setCurrentView`. O método `project` dessa classe faz a projeção de uma determinada coordenada na OpenGL pela coordenada a ser usada para desenhar os textos na classe `LabelMaker`.

A classe `MatrixGrabber` quando acionada, obtém as últimas transformações armazenadas pelo primeiro conjunto de classes desse pacote e guarda em seus vetores internos. Os três métodos dessa classe são utilizados pela classe `Projector` quando for necessário guardar as transformações da OpenGL. Considera-se que o objeto `GL10` é uma implementação da classe `MatrixTrackingGL` e através dele obtém-se as matrizes de transformação.

No último conjunto de classes desse pacote, a classe `LabelMaker` é responsável por criar uma textura para cada texto e desenhá-la na OpenGL. A estratégia para desenhar o texto é a de primeiro desenhá-lo em um objeto de `Canvas`, através dele criar uma textura e adicioná-la ao `GL10`. Os métodos `beginAdding`, `add` e `endAdding` devem ser chamados nessa ordem para adicionar os textos a serem desenhados. O métodos `beginDrawing`, `draw` e `endDrawing`

são chamados para que os textos já adicionados sejam desenhados no `GL10`. A posição *x* e *y* do método `draw` deve ser obtida através do retorno do método `Projector.project`.

Por último, a classe `ASCIISprite` utiliza os recursos da classe `LabelMaker` para desenhar todos os caracteres da tabela *American Standard Code for Information Interchange* (ASCII), cada um como uma textura em separado de forma a junta-los quando for necessário montar textos dinâmicos. Essa classe não está disponível na referência Google (2010m) e sua necessidade se deu pelo fato de que os pontos de interesse são dinâmicos, portanto seus nomes devem ser adicionados e removidos com freqüência da classe `LabelMaker`.

3.3.2.4 Pacote `br.furb.ra.view`

As classes desse pacote são responsáveis por desenhar a primeira e a última camada da tela. A Figura 17 mostra o diagrama de classes desse pacote.

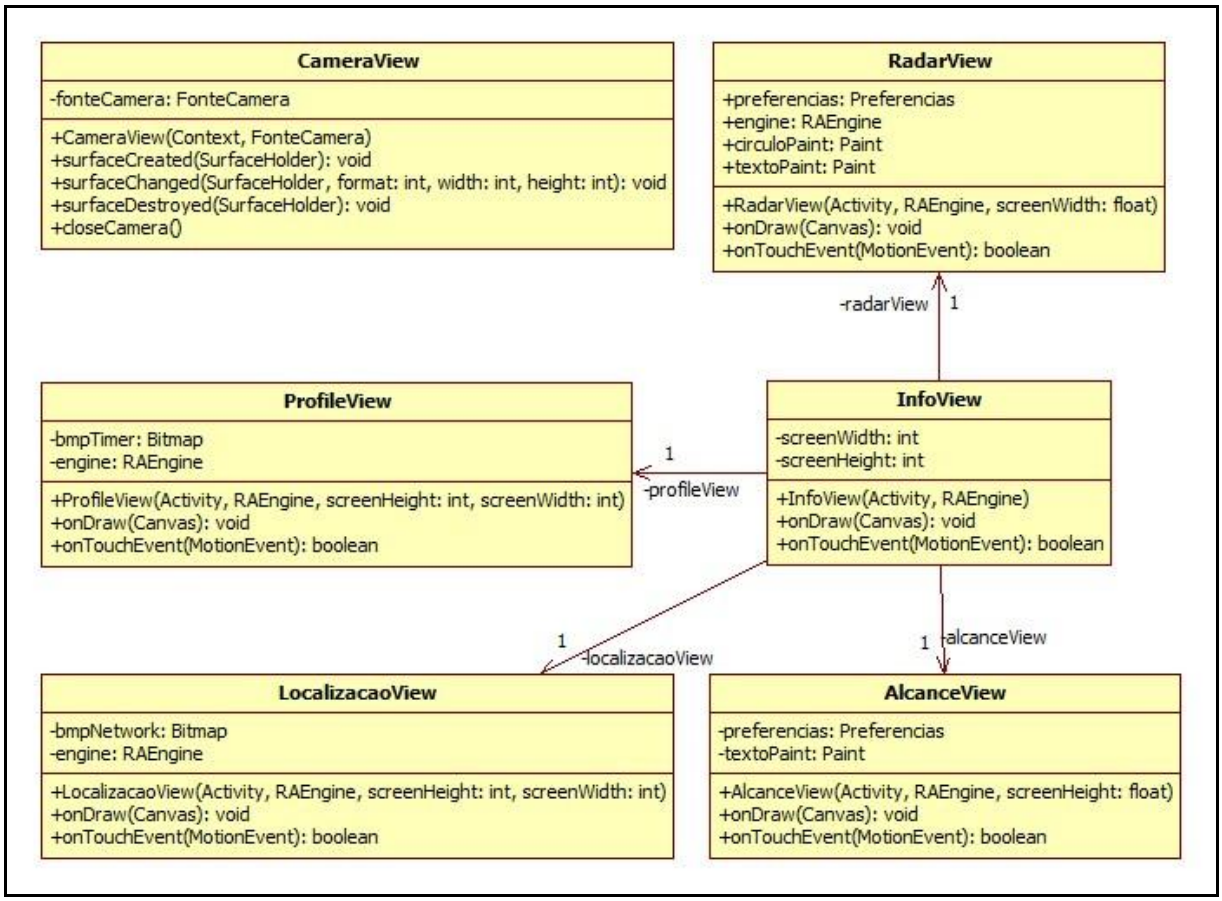


Figura 17 - Pacote `br.furb.ra.view`

A classe `CameraView` é instanciada pela classe `RAActivity` e seu objetivo é gerenciar um objeto de `FonteCamera` e apresentar as imagens da câmera na primeira camada da tela.

Seus métodos `surfaceCreated`, `surfaceChanged` e `surfaceDestroyed` são chamados pela API do Android quando ocorre alguma mudança no estado da tela e a implementação de cada um deles gerencia a fonte de câmera de acordo com a mudança ocorrida.

A classe `InfoView` também é instanciada pela classe `RAActivity` e seu objetivo é desenhar a terceira camada da tela, a camada das ferramentas. Sua classe base é `View` e a partir dela foram sobrescritos os métodos `onDraw` e `onTouchEvent`. Essa classe instancia e gerencia os objetos que farão parte da terceira camada de tela, delegando o desenho de cada objeto através do método `onDraw` e delegando o tratamento do toque através do método `onTouchEvent`.

A primeira classe gerenciada pela classe `InfoView` é a classe `RadarView` e sua responsabilidade é desenhar um radar no canto direito da tela. No método `onDraw` é desenhado um círculo cinza semi-translúcido contendo pequenos pontos brancos que representam os pontos de interesse. Também são desenhados os quatro pontos cardeais na sua respectiva direção de acordo com o valor de *azimuth* obtido pela *engine*. O método `onTouchEvent` é chamado pela API do Android sempre que houver um toque na tela, em sua implementação é verificado se o toque foi dentro do radar para que seja aberta a tela de configuração do seu raio de alcance.

As classes `ProfileView` e `LocalizacaoView` possuem implementações semelhantes, ambas desenham um ícone na tela e tratam o evento de toque. A classe `ProfileView` desenha o ícone de um cronômetro na parte inferior central da tela e trata o toque nesse ícone mostrando os *Frames Per Second* (FPS) obtidos pela aplicação. Já a classe `LocalizacaoView` desenha um ícone no canto direito inferior da tela e trata o toque nesse ícone mostrando a latitude, a longitude, a altitude, a fonte de localização geográfica utilizada e sua precisão.

Por último, a classe `AlcanceView` desenha um texto no canto esquerdo inferior da tela e trata o evento de toque permitindo configurar o valor do alcance em que serão mostradas as setas e os painéis dos pontos de interesse.

3.3.2.5 Pacote `br.furb.ra.poi`

Nesse pacote estão as classes responsáveis por representar um ponto de interesse e por fornecer os pontos de interesse através de uma fonte fixa ou pela *web*. A Figura 18 mostra o diagrama de classes do pacote.

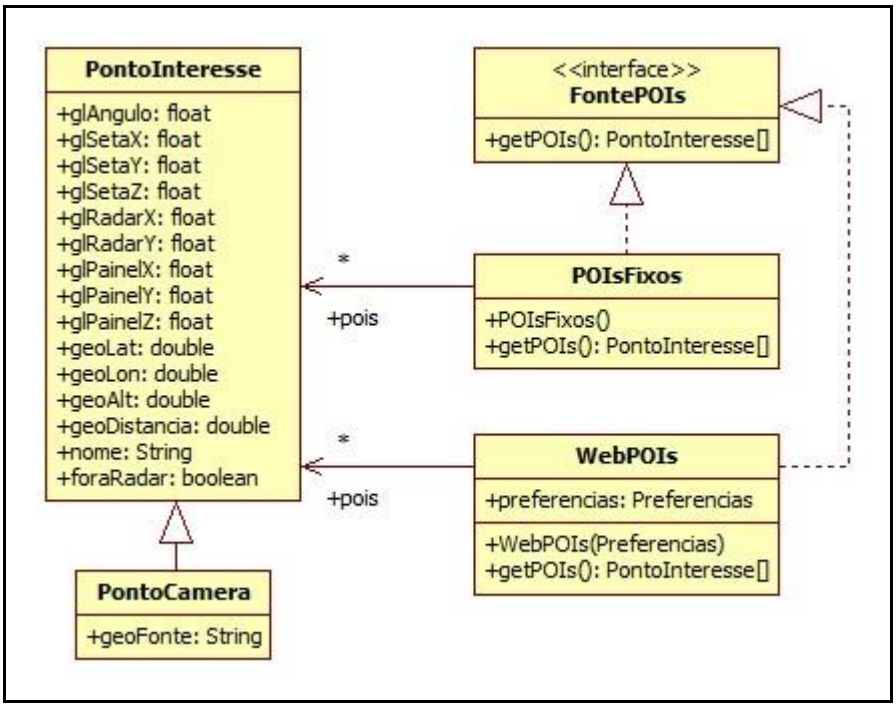


Figura 18 - Pacote `br.furb.ra.poi`

A classe `PontoInteresse` guarda todas as informações relativas a um ponto de interesse a ser desenhado na OpenGL. Os atributos `geoLat`, `geoLon` e `geoAlt` são preenchidos com os valores da coordenada geográfica que se encontra o ponto de interesse. O atributo `geoDistancia` é calculado pela `RAEngine` com base na distância do ponto de interesse para o dispositivo. São calculados também os atributos `glAngulo`, `glSetaX`, `glSetaY` e `glSetaZ` responsáveis por determinar a posição da seta que aponta para o ponto de interesse, os atributos `glRadarX` e `glRadarY` responsáveis por posicionar o ponto de interesse no radar e os atributos `glPainelX`, `glPainelY` e `glPainelZ` responsáveis por posicionar o painel do ponto de interesse.

A classe `PontoCamera` é filha da classe `PontoInteresse`, adicionando o atributo `geoFonte` que guarda o nome da última fonte de localização geográfica válida obtida para o dispositivo.

A interface `FontePOIs` define o comportamento de uma fonte de pontos de interesse que através do método `getPOIs` deve retornar a lista dos pontos de interesse. Sua primeira implementação é a classe `POIsFixos` que fornece alguns pontos de interesse fixos definidos na própria implementação da classe. Já a classe `WebPOIs` tem a responsabilidade de obter os pontos de interesse a partir de uma conexão.

3.3.2.6 Pacote `br.furb.ra.hardware`

As classes que compõem esse pacote fornecem todos os recursos de interação com o *hardware* do dispositivo ou com os simuladores. A Figura 19 mostra o diagrama de classes dos recursos para a API de sensores desse pacote.

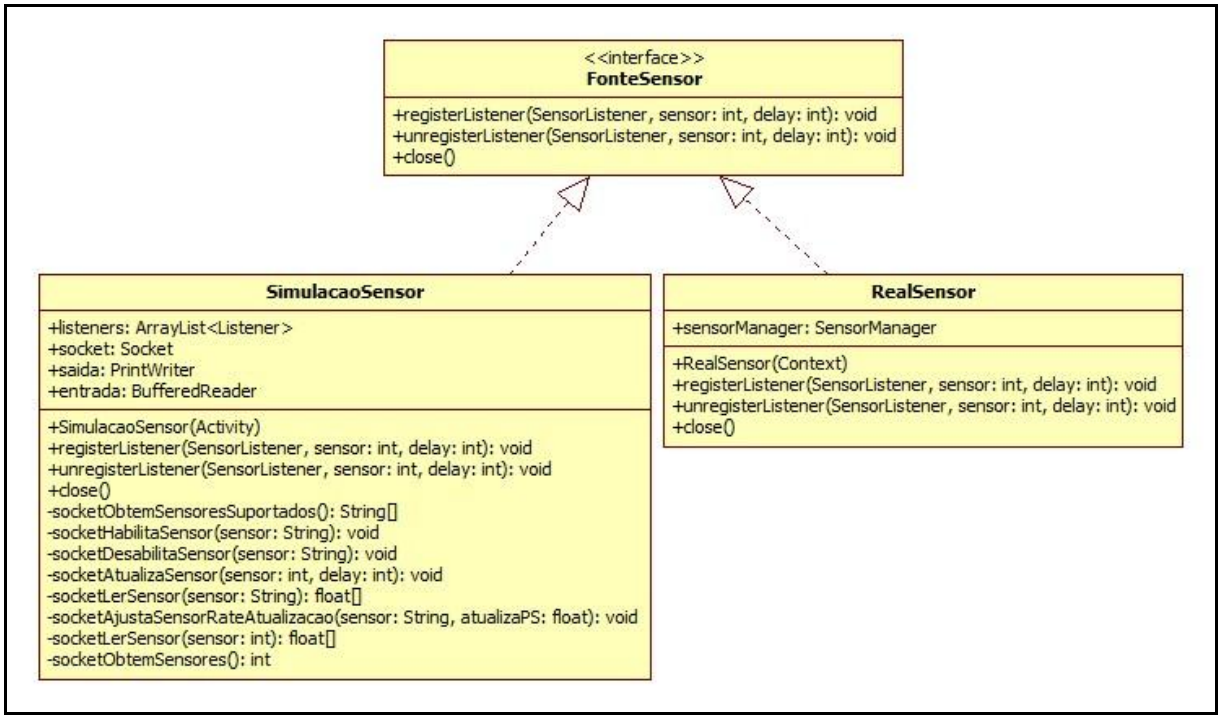


Figura 19 - API de sensores

A interface `FonteSensor` representa uma fonte que fornece o valor dos sensores. Através dos métodos `registerListener` e `unregisterListener` é possível registrar e desregistrar observadores para as mudanças nos sensores. O método `close` é usado para fechar a comunicação com a fonte de sensor.

A classe `RealSensor` utiliza a API do Android através de delegação, adicionando ou removendo os observadores diretamente na classe `SensorManager` do Android. Já a classe `SimulacaoSensor` é responsável por comunicar-se via *socket* com um servidor para obter os valores dos sensores.

A Figura 20 mostra um diagrama das classes responsáveis por prover os recursos de câmera.

A interface `FonteCamera` representa uma fonte que fornece as imagens da câmera e as disponibiliza no objeto `SurfaceHolder`. Os métodos `onCreated`, `onChanged` e `onDestroyed` definem o ciclo de vida da comunicação com a câmera para que seja possível ligá-la ou desligá-la quando necessário.

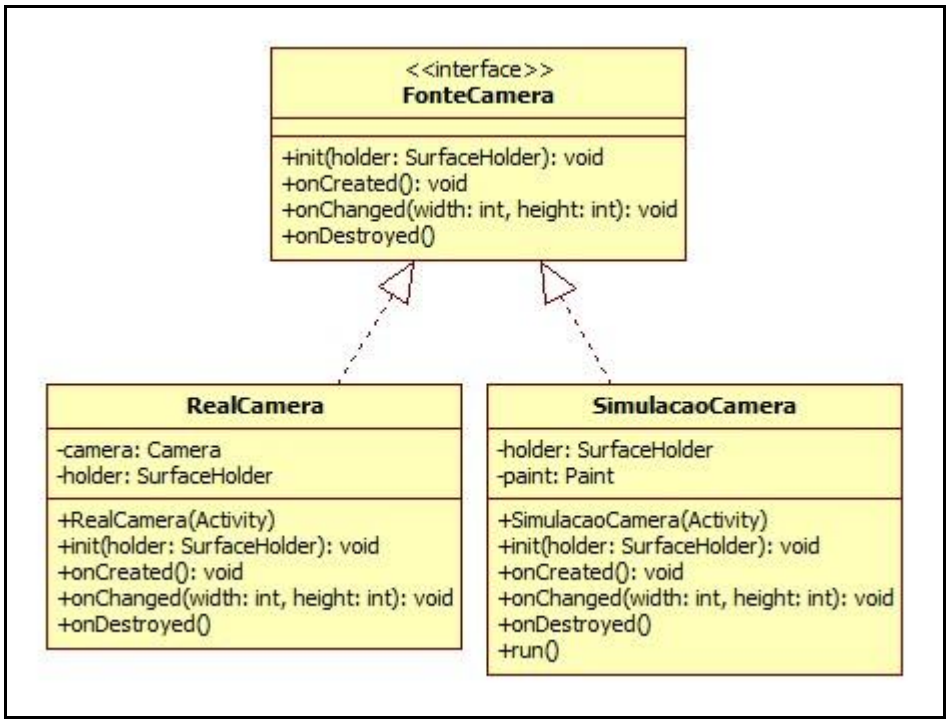


Figura 20 - API de câmera

A classe `RealCamera` utiliza a classe `Camera` disponível na API do Android para receber as imagens da câmera e disponibilizá-las no objeto `SurfaceHolder`. Já a classe `SimulacaoCamera` é responsável por comunicar-se via *socket* com um servidor para obter as imagens da câmera.

Por último a classe `SimulacaoGPS`, representada no diagrama da Figura 21, utiliza os recursos disponíveis na API de localização do próprio Android para fazer a simulação da localização geográfica, conforme descrito na sessão 2.2.6.

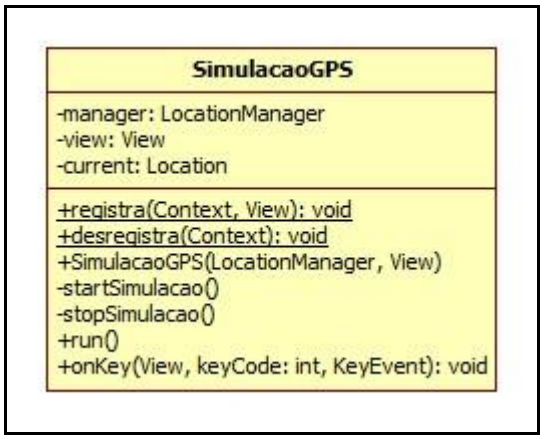


Figura 21 - Simulação da localização

3.3.2.7 Pacote `br.furb.ra.config`

O pacote `br.furb.ra.config` possui classes responsáveis por fornecer recursos para gravar as configurações definidas pelo usuário de forma que não sejam perdidas ao sair da aplicação. A Figura 22 mostra o diagrama de classes do pacote.

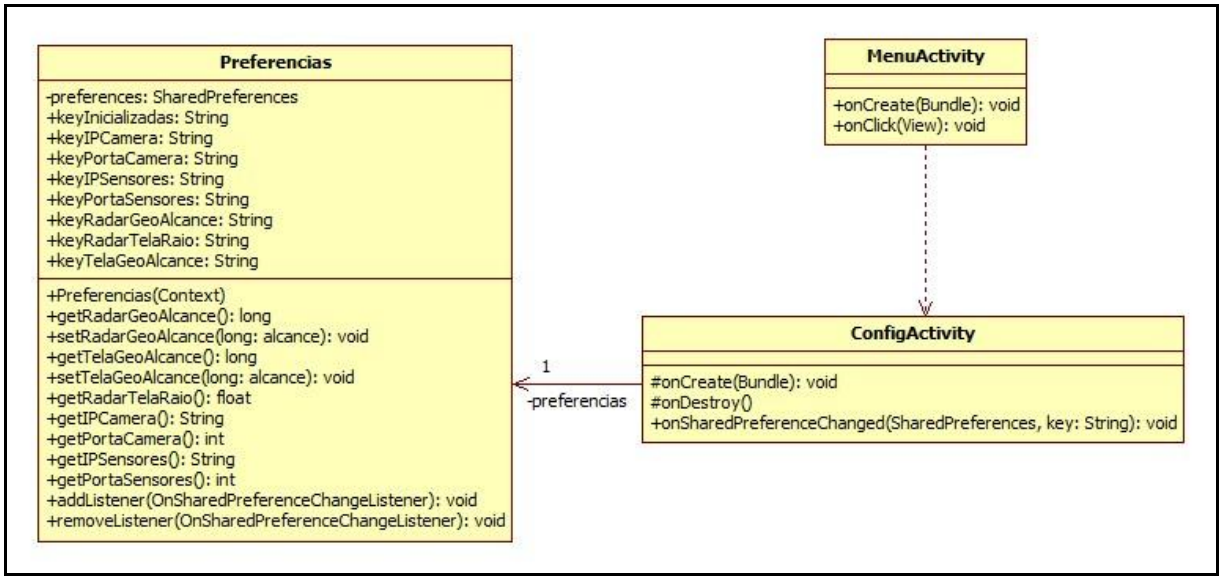


Figura 22 - Pacote `br.furb.ra.config`

A classe `MenuActivity` é a primeira classe instanciada na aplicação, sua responsabilidade é mostrar um menu que permite entrar nas configurações ou entrar no aplicativo de realidade aumentada. Sua implementação tem por classe base a classe `Activity` do Android. É através do mecanismo de intenções que essa classe abre as duas outras atividades da aplicação, sendo elas `ConfigActivity` e `RAActivity`.

A classe `ConfigActivity` é chamada pela `MenuActivity` assim que o botão de configuração for pressionado. Sua implementação tem por classe base `PreferenceActivity` do Android que cuida do ciclo de vida de uma atividade de preferências. Sua responsabilidade é fornecer campos para edição das configurações da aplicação.

Na classe `Preferences`, o atributo `preferences` é da classe `SharedPreferences` disponível na API do Android para gravar em arquivo pequenas configurações. Através dos métodos `getRadarGeoAlcance`, `getTelaGeoAlcance`, `getRadarTelaRaio` `getIPCamera`, `getPortaCamera`, `getIPSensores` e `getPortaSensores` é possível obter valores das preferências. Através dos métodos `setRadarGeoAlcance`, `setTelaGeoAlcance` é possível ajustar valores nas preferências. O método `addListener` permite adicionar observadores de mudanças nas preferências e o método `removeListener` permite removê-los.

3.3.2.8 Pacote `br.furb.ra.util`

Neste pacote estão as classes utilitárias da aplicação cujo diagrama está representado na Figura 23.

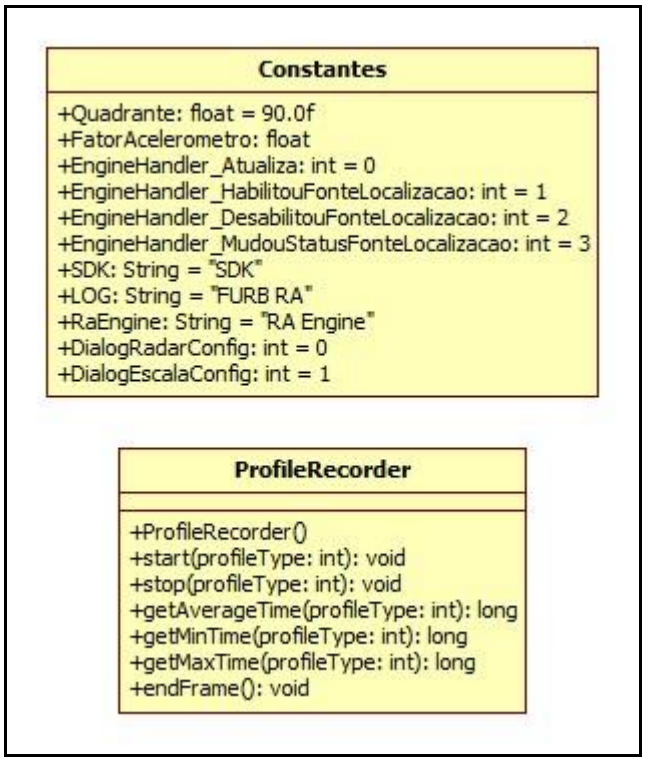


Figura 23 - Pacote `br.furb.ra.util`

A classe `Constantes` define uma série de constantes utilizadas ao longo da aplicação, tanto para diferenciar tipos de eventos como `EngineHandler_Atualiza` quanto para definir nome de *log* ou de *thread* como os atributos `LOG` e `RaEngine`.

Já a classe `ProfileRecorder` é responsável por gravar medições de desempenho, definir as médias, os mínimos e máximos para cada medição. Os métodos `start` e `stop` iniciam e finalizam determinada medição. Os métodos `getAverage`, `getMinTime` e `getMaxTime` retornam a média, o mínimo e o máximo tempo para determinada medição.

3.3.2.9 Fornecedor de pontos de interesse

Para fornecer os pontos de interesse a partir de um servidor *web* foi criada a classe `FontePOIsServlet`. Essa classe não pertence à aplicação de realidade aumentada para o dispositivo Android, ela faz parte de um projeto *web* e deve rodar em um servidor. Seus

atributos e métodos são demonstrados no diagrama da Figura 24.

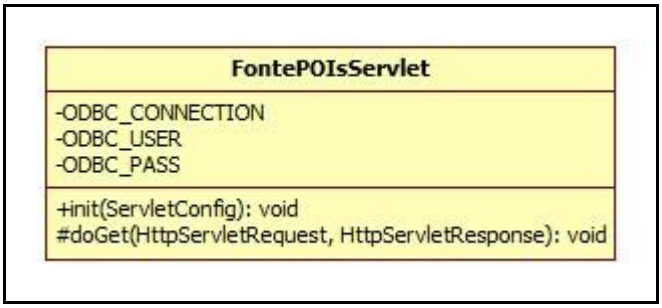


Figura 24 - Fornecedor de pontos de interesse

Seu principal método é `doGet`, sobrescrito da classe base `HttpServlet`, que é responsável por receber uma requisição através do objeto `HttpServletRequest` e enviar os pontos de interesse na resposta através do objeto `HttpServletResponse`. Os pontos de interesse nesse método retornados são obtidos através de uma conexão com um banco de dados.

3.3.3 Diagrama de seqüência

O diagrama de seqüência da Figura 25 mostra a interação do `Usuário` com a aplicação de realidade aumentada no desenho de setas e painéis dos casos de uso `UC01` e `UC02`.

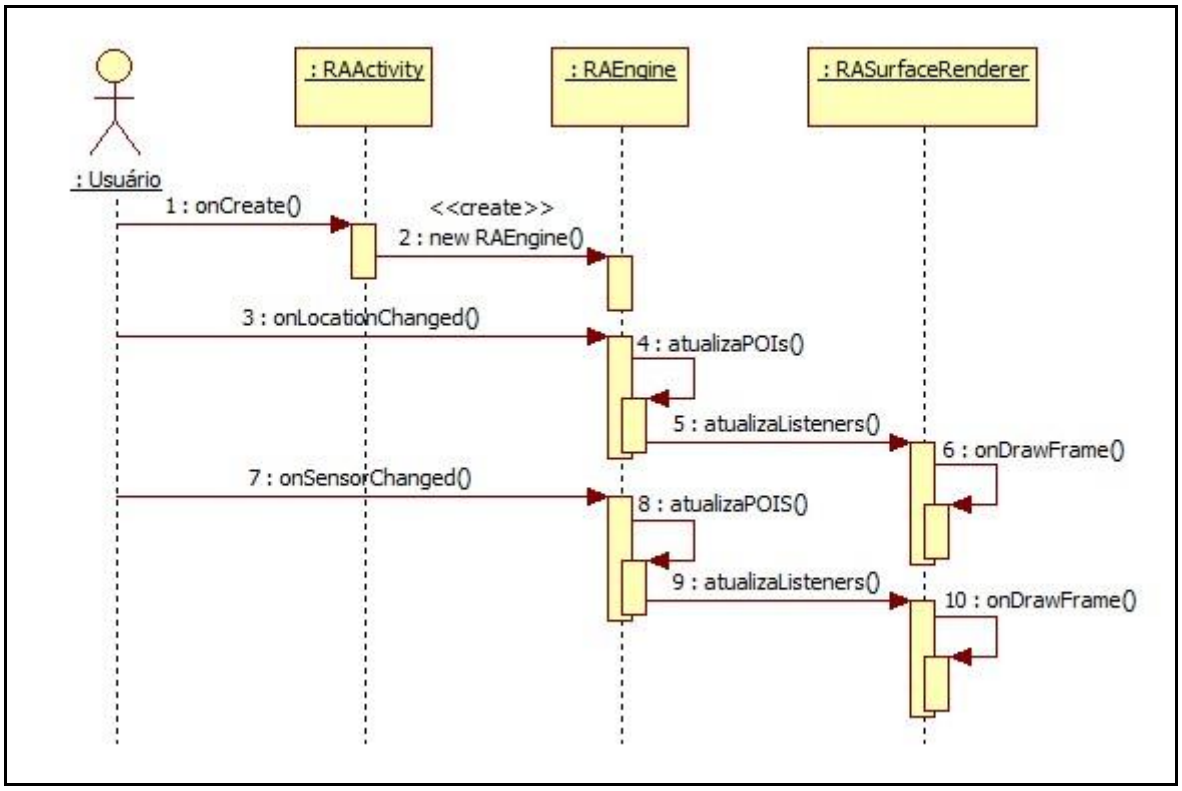


Figura 25 - Diagrama de seqüencia da aplicação de realidade aumentada

3.4 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas na implementação, bem como detalhes das principais classes e rotinas implementadas durante o desenvolvimento da aplicação.

3.4.1 Técnicas e ferramentas utilizadas

O desenvolvimento da aplicação de realidade aumentada foi feito na linguagem Java com a API de desenvolvimento do Android na versão 2.2, também chamada de Froyo. O ambiente de desenvolvimento utilizado foi o Eclipse com o conjunto de *plugins* do Android *Development Tools* (ADT), que possui uma série de recursos para criação, execução e depuração de aplicações Android de forma a facilitar o desenvolvimento. Para a execução e a depuração da aplicação também foi utilizado um dispositivo *smartphone* da fabricante HTC chamado HTC Desire que possui o sistema operacional Android Froyo.

O dispositivo HTC Desire possui processador Qualcomm Snapdragon QSD8250 com 1 *Giga Hertz* (GHz), sua GPU é AMD Z430, com memória de 576 *Mega Bytes* (MB) de RAM, 512 MB de *Read Only Memory* (ROM) e resolução de 480x800 (MOBILE TECH WORLD, 2010).

Para o servidor de pontos de interesse foi utilizada a especificação JEE na definição de um `Servlet` que recebe requisições no protocolo *HyperText Transfer Protocol* (HTTP). Os pontos de interesse são obtidos a partir de um banco de dados MySQL na versão 5.2 através da interface Java *DataBase Connectivity* (JDBC). O servidor utilizado para rodar a aplicação *web* foi o Apache Tomcat na versão 7.0.4.

3.4.2 A *engine*

De todas as classes da aplicação, pode-se dizer que a `RAEngine` é a que mais acumula funções e agrega recursos. A responsabilidade dessa classe é de manter atualizados os pontos de interesse e a localização da câmera, tanto no que diz respeito à realidade (coordenadas geográficas e posição da bússola) quanto no que diz respeito ao aumento de realidade (posição das setas e dos painéis na tela).

Logo ao ser instanciado um objeto da classe `RAEngine`, esse passa a ser observador de mudanças (Quadro 9) nas coordenadas geográficas, no sensor de acelerômetro, no sensor de orientação e nas preferências.

```
public RAEngine(/*Lista de parâmetros omitida*/) {
	// (...)
	// Escuta mudanças nas coordenadas geográficas.
	fonteLocalizacao.requestLocationUpdates(LocationManager.GPS_PROVIDER, 250, 1.0f,
this);
	fonteLocalizacao.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 250, 1.0f,
this);

	// Escuta mudanças na orientação e no acelerômetro.
	fonteSensor.registerListener(this, SensorManager.SENSOR_ORIENTATION,
SensorManager.SENSOR_DELAY_UI);
	fonteSensor.registerListener(this, SensorManager.SENSOR_ACCELEROMETER,
SensorManager.SENSOR_DELAY_UI);

	// Escuta mudanças nas preferências.
	preferencias.addListener(this);
}
```

Quadro 9 - Observação das mudanças no dispositivo

Ao receber a mensagem de que houve uma mudança na localização geográfica do dispositivo (Quadro 10), a *engine* obtém os novos pontos de interesse e guarda a nova localização no atributo `pontoCamera`. Por último a variável `dirty` é ajustada para `true` avisando que a *engine* entrou no estado `Desatualizado`.

```
public void onLocationChanged(final Location location) {
	this.precisaoLocalizacao = location.getAccuracy();

	// Obtém as novas lat/lon/alt do dispositivo.
	final double cameraGeoLat = location.getLatitude();
	final double cameraGeoLon = location.getLongitude();
	final double cameraGeoAlt = location.getAltitude();

	// Obtém os novos POIs.
	this.pois = this.fontePOIs.getPOIs();

	// Atualiza no modelo.
	this.pontoCamera.geoLat = cameraGeoLat;
	this.pontoCamera.geoLon = cameraGeoLon;
	this.pontoCamera.geoAlt = cameraGeoAlt;
	this.pontoCamera.geoFonte = location.getProvider();

	// Avisa que está desatualizado.
	synchronized (this) {
		this.dirty = true;
	}
}
```

Quadro 10 – Guardar a nova localização

Quando ocorre alguma mudança no sensor de orientação (Quadro 11), a *engine* calcula o ângulo do *azimuth* com base no norte verdadeiro da Terra, ajustando o ângulo do norte magnético através da classe `GeomagneticField`. A atualização do *azimuth* somente ocorre quando há uma mudança no seu valor decimal.

```
public void onSensorChanged(final int sensor, final float[] valores) {
	if (sensor == SensorManager.SENSOR_ORIENTATION) {
		final float orientationX = valores[0];
		final float lat = (float) this.pontoCamera.geoLat;
		final float lon = (float) this.pontoCamera.geoLon;
		final float alt = (float) this.pontoCamera.geoAlt;

		// Ajusta com o valor do norte verdadeiro.
		final GeomagneticField geoField = new GeomagneticField(lat, lon, alt,
System.currentTimeMillis());
		final float azimuth = orientationX + geoField.getDeclination();

		// Verifica se vai atualizar.
		final int newAzimuth = (int) azimuth;
		final int oldAzimuth = (int) this.azimuth;
		if (oldAzimuth != newAzimuth) {
			this.azimuth = azimuth;
			synchronized (this) {
				this.dirty = true;
			}
		}
		// (...)
	}
}
```

Quadro 11 – Guardar a nova orientação

Já quando ocorre alguma mudança no acelerômetro (Quadro 12), o ângulo de inclinação deve ser calculado com base no valor da gravidade. Os valores retornados pelo acelerômetro variam de `-9.80665f` a `+9.80665f`, quando a movimentação do dispositivo se dá em um ângulo de 180º. Em outras palavras, a cada 90º de movimentação do dispositivo (seja ela percorrendo o *azimuth*, o *pitch* ou o *roll*), o valor retornado pelo acelerômetro será ente zero e o valor da gravidade (podendo ser positivo ou negativo).

```
public void onSensorChanged(final int sensor, final float[] valores) {
	// (...)
	else if (sensor == SensorManager.SENSOR_ACCELEROMETER) {
		final float acelerometroY = valores[1];
		final float acelerometroZ = valores[2];

		// Obtém a orientação real no y.
		float newAcelerometroZ = Constantes.FatorAcelerometro * acelerometroZ -
Constantes.Quadrante;
		if (acelerometroY > 0) {
			newAcelerometroZ *= -1;
		}
		if (newAcelerometroZ != this.roll) {
			this.roll = newAcelerometroZ;
			synchronized (this) {
				this.dirty = true;
			}
		}
	}
}
```

Quadro 12 - Guardar a movimentação no acelerômetro

O método que confere à classe `RAEngine` o título de *engine* chama-se `atualizaPOIs`. Através desse método são realizados todos os cálculos de posicionamento, angulação e validação dos pontos de interesse. Dada sua complexidade, a implementação foi selecionada e dividida em alguns quadros.

O Quadro 13 mostra parte do código principal do método e um algoritmo para facilitar o entendimento do comportamento do método. Os passos do algoritmo estão descritos nos

próximos parágrafos.

```
private void atualizaPOIs() {
        // (...)
        final float radarGeoAlcance = this.preferencias.getRadarGeoAlcance();
        final float radarTelaRaio = this.preferencias.getRadarTelaRaio();
        final long telaGeoAlcance = this.preferencias.getTelaGeoAlcance();

        final int poisLength = this.pois.length;
        for (int index = 0; index < poisLength; index++) {
                final PontoInteresse poi = this.pois[index];

                // Calcula a distância do POI para o dispositivo.
                // Calcula a localização do POI no radar.
                // Calcula a localização e o ângulo da seta do POI na OpenGL.
                // Calcula a localização do painel do POI na OpenGL.
        }

        // (...)
}
```

Quadro 13 - Algoritmo de atualização dos pontos de interesse

A distância entre dois pontos na Terra com base em suas latitudes e longitudes pode ser obtida utilizando o Teorema de Pitágoras quando essa distância for menor do que 20 quilômetros (GIS FAQ, 2010). O erro obtido por esse cálculo é menor do que 30 metros (GIS FAQ 2010). Assumindo que a Terra possui um raio perfeito é possível utilizar a fórmula de Haversine, obtendo resultados exatos para uma elipse perfeita (GIS FAQ, 2010).

Por outro lado a *National Geospatial-Intelligence Agency* (NGA, 2010) desenvolve e mantém o Sistema Mundial Geodésico, ou melhor conhecido por *World Geodetic System* (WGS) que é o padrão utilizado para definir a forma e o tamanho irregulares da Terra. A última revisão desse padrão chama-se WGS84 datada de 1984 e será válida até o ano de 2010 (NGA, 2010).

A classe `Location` do Android, através do método `distanceTo`, utiliza o padrão WGS84 para definir a distância entre dois pontos na Terra considerando suas latitudes, longitudes, altitudes e também a data atual. Esse método está sendo utilizado para obtenção da distância entre cada ponto de interesse e o dispositivo. Somente são retornados os pontos de interesse cuja distância estiver dentro do alcance configurado para a aplicação, conforme demonstrado no Quadro 14. O valor da distância é armazenado no atributo `geoDistancia` no objeto do ponto de interesse.

```
final Location camLocation = new Location(RAEngine.class.getName());
camLocation.setTime(System.currentTimeMillis());
camLocation.setLatitude(cameraGeoLat);
camLocation.setLongitude(cameraGeoLon);
camLocation.setAltitude(cameraGeoAlt);
// (...)

// Verifica se está numa distância aceitável
final Location poiLocation = new Location(RAEngine.class.getName());
poiLocation.setTime(System.currentTimeMillis());
poiLocation.setLatitude(poi.geoLat);
poiLocation.setLongitude(poi.geoLon);
poiLocation.setAltitude(poi.geoAlt);

// Obtém a distância (WGS84).
poi.geoDistancia = camLocation.distanceTo(poiLocation);

// Guarda se o POI está muito longe.
poi.foraTela = Math.abs(poi.geoDistancia) > telaGeoAlcance;
// (...)
```

Quadro 14 - Obtendo a distância entre um ponto de interesse e o dispositivo

O Quadro 15 mostra como é calculada a posição no radar de um ponto de interesse e também a verificação feita para desenhar apenas os pontos de interesse que estão dentro da configuração do alcance do radar.

```
// Verifica se está muito distante a ponto de não pintar no radar.
final boolean foraRadar = Math.abs(poi.geoDistancia) >= radarGeoAlcance;

// Guarda a posição do ponto em miniatura dentro do radar.
if (foraRadar) {
        poi.foraRadar = true;
        poi.glRadarX = Float.NaN;
        poi.glRadarY = Float.NaN;
}
else {
        poi.foraRadar = false;
        final double telaDistancia = poi.geoDistancia / radarGeoAlcance * radarTelaRaio;
        poi.glRadarX = (float) (telaDistancia * anguloSin);
        poi.glRadarY = (float) (telaDistancia * anguloCos);
}
```

Quadro 15 - Calcular a posição do ponto de interesse no radar

A posição da seta e a posição do painel do ponto de interesse são calculadas a partir do seno e do cosseno do ângulo que há entre o ponto de interesse e o dispositivo, esses valores são armazenados nos atributos `glSetaX`, `glSetaY`, `glPainelX` e `glPainelY` do objeto do ponto de interesse. Também o ângulo é importante para que a seta seja rotacionada e o painel seja posicionado de forma equivalente, por isso é armazenado no atributo `glAngulo`, conforme demonstrado no Quadro 16.

```
// Guarda o ângulo pois vai usar na seta e no painel.
poi.glAngulo = (float) Math.toDegrees(rAngulo);

// Guarda a posição da seta no gl.
poi.glSetaX = (float) (glDiametroCamera * anguloSin);
poi.glSetaY = (float) (glDiametroCamera * anguloCos);

// Guarda a posição do painel no gl.
poi.glPainelX = (float) (RAEngine.DistanciaPaineis * anguloSin);
poi.glPainelY = (float) (RAEngine.DistanciaPaineis * anguloCos);
```

Quadro 16 - Calcular a posição e o ângulo da seta e do painel do ponto de interesse

O método que confere à *engine* o comportamento de uma *thread* chama-se `run` e é através dele que os estados (vide Figura 14) são implementados. Conforme o Quadro 17, a

thread irá executar até que o atributo `threadRunning` seja falso, tornando este o atributo responsável pelo estado `Executando`. Em primeiro momento é feita a verificação se a aplicação não está no estado `Pausado` através do atributo `threadPaused`, fazendo a *thread* dormir por alguns milissegundos até verificar novamente se está pausada. Depois verifica-se o estado `Desatualizado` através do atributo `dirty`, atualizando os pontos de interesse pelo método `atualizaPOIs` e voltando ao estado `Executando`. Por último a *thread* dorme por alguns milissegundos, entrando no estado `Dormindo` e voltando ao estado `Executando` ao término desse tempo.

```
public void run() {
	while (this.threadRunning) { // Estado "Executando"

		// Não atualiza nada enquanto a aplicação estiver parada.
		while (this.threadPaused) { // Estado "Pausado"
			try {
				Thread.sleep(100);
			} catch (final InterruptedException e) {	}
		}

		// Se estiver desatualizada, atualiza.
		synchronized (this) {
			if (this.dirty) { // Estado "Desatualizado"
				this.atualizaPOIs();
				this.atualizaListeners();
				this.dirty = false; // Estado "Executando"
			}
		}

		// Dorme por alguns ms.
		try {
			Thread.sleep(100); // Estado "Dormindo"
		} catch (final InterruptedException e) {	}
	}
}
```

Quadro 17 - Execução da *thread* da *engine*

Quando a aplicação é finalizada, o método `onDestroy` (Quadro 18) é chamado pela classe `RAActivity`. É através dele que a *engine* para de observar mudanças na fonte de localização, na fonte de sensor e nas preferências, também e desligada a *thread* ao término desse método.

```
public void onDestroy() {
	// Remove a escuta pelas coordenadas geográficas.
	this.fonteLocalizacao.removeUpdates(this);

	// Remove a escuta pela bússola.
	this.fonteSensor.unregisterListener(this,SensorManager.SENSOR_ORIENTATION);
	this.fonteSensor.unregisterListener(this,SensorManager.SENSOR_ACCELEROMETER);
	this.fonteSensor.close();

	// Para de escutar pelas preferências.
	this.preferencias.removeListener(this);

	// Desliga a thread.
	synchronized (this) { // Estado "Finalizado"
		this.threadPaused = false; // Despausa caso esteja.
		this.dirty = false; // Não calcula mais.
		this.threadRunning = false; // Desliga a thread.
	}
}
```

Quadro 18 - Finalizando a *engine*

3.4.3 As telas da aplicação

A aplicação possui três telas, uma que mostra a realidade aumentada, outra para configurar os parâmetros da realidade aumentada e uma terceira que é a tela inicial, permitindo acesso às outras duas.

A primeira tela é implementada pela classe `MenuActivity`, sua criação ocorre pela API do Android e, por ser uma `Activity`, possui o ciclo de vida determinado pelo Android. As demais telas são invocadas pelo mecanismo de intenções e serão chamadas quando os botões do menu forem pressionados. O Quadro 19 mostra a implementação dos dois métodos dessa classe, o primeiro criando os componentes de tela e o segundo utilizando as intenções para abrir as demais telas.

```
protected void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        this.setContentView(R.layout.menu);
        this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        ((Button) this.findViewById(R.id.config)).setOnClickListener(this);
        ((Button) this.findViewById(R.id.ra)).setOnClickListener(this);
}
public void onClick(final View v) {
        // Abre a atividade de configuração e aguarda o retorno.
        if (v.getId() == R.id.config) {
                this.startActivity(new Intent(this, ConfigActivity.class));
        }
        // Abre a atividade de Realidade Aumentada.
        else if (v.getId() == R.id.ra) {
                this.startActivity(new Intent(this, RAActivity.class));
        }
}
```

Quadro 19 - Implementação do menu

A tela de configurações chama-se `ConfigActivity` (Quadro 20) e sua classe base é `PreferenceActivity` definida pela API do Android para simplificar telas de preferências. As preferências da aplicação estão definidas no arquivo `preferências.xml` e são inseridas na tela através da chamada `addPreferencesFromResource`. A classe base do Android monta uma tela compatível com as preferências definidas no arquivo XML em questão.

```
public class ConfigActivity extends PreferenceActivity {
        // (...)
        protected void onCreate(final Bundle savedInstanceState) {
                this.requestWindowFeature(Window.FEATURE_NO_TITLE);
                super.onCreate(savedInstanceState);

                // Adiciona as preferências.
                this.addPreferencesFromResource(R.xml.preferencias);

                // Inicializa as preferências com valores default.
                this.preferencias = new Preferencias(this);
                this.preferencias.addListener(this);

                // (...)
        }
        // (...)
}
```

Quadro 20 - Implementação das configurações

A tela de realidade aumentada é implementada pela classe `RAActivity` e é a única que não segue fielmente o ciclo de vida proposto pelo Android para suas aplicações. Devido a uma limitação do Android no mecanismo de sobreposição de camadas, essa tela ao entrar no estado `Pausado` através do método `onPause` (Quadro 21), força a finalização da tela para que quando ela for reaberta, que seja recarregada.

```
protected void onPause() {
        this.raView.onPause();
        this.raEngine.onPause();
        super.onPause();
        /**
         * Não existe como sair e voltar pra aplicação sem finalizá-la. Isso aqui é foi feito
para que ao sair com o botão Home a aplicação seja finalizada de forma forçada. Sem isso ao
iniciar a aplicação novamente o OpenGL não conseguiria desenhar nada na tela (de alguma forma
a câmera se sobrepõe ao GL).

Para mais detalhes leia a última mensagem da thread:
http://groups.google.com/group/android-
developers/browse_thread/thread/9c335071c7919d80/827167f6a3dc08ee?pli=1
         */
        this.finish();
}
```

Quadro 21 - Sobrescrita do método `onPause`

3.4.4 As camadas da tela de realidade aumentada

A interface gráfica da tela de realidade aumentada foi implementada de forma a possuir de três camadas sobrepostas conforme especificado na sessão 3.3.2.1. A criação das camadas e a sua sobreposição (Quadro 22) ocorre no momento em que a aplicação é criada, no método `onCreate` da classe `RAActivity`.

```
protected void onCreate(final Bundle savedInstanceState) {
        // (...)
        // Coloca landscape.
        this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        // Coloca a tela em full screen.
        this.requestWindowFeature(Window.FEATURE_NO_TITLE);
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
WindowManager.LayoutParams.FLAG_FULLSCREEN);

        // Não permite que o sistema desligue a tela.
        this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        // (...)
        // Cria as camadas da tela.
        this.raView = new RASurfaceView(this, this.raEngine);
        this.cameraView = new CameraView(this, fonteCamera);
        final InfoView infoView = new InfoView(this, this.raEngine);

        // Adiciona as camadas na tela.
        final FrameLayout frame = new FrameLayout(this);
        frame.addView(this.raView);
        frame.addView(this.cameraView);
        frame.addView(infoView);
        this.setContentView(frame);
        // (...)
}
```

Quadro 22 - Criação das camadas da tela

A criação das camadas ocorre ao instanciar objetos das classes `RASurfaceView`, `CameraView` e `InfoView`. Já a sobreposição ocorre ao adicionar os objetos na `activity` do Android através dos métodos `setContentView` e `addView`. No início do método a aplicação é ajustada para a orientação de tela *landscape* e faz-se a requisição para que ela ocupe a tela toda.

A primeira camada da tela, e por conseqüência a que está embaixo das demais, é a câmera implementada pela classe `CameraView`. Essa classe ao ser instanciada (Quadro 23) recebe uma fonte de câmera, objeto da interface `FonteCamera`, que através do polimorfismo pode ser tanto uma fonte de câmera vinda do simulador quanto vinda do dispositivo. Ainda no construtor é obtido o objeto de `SurfaceHolder` para adicionar-se como um observador das mudanças que ocorrem no ciclo de vida de uma tela, criação, alteração e finalização. Por último a fonte de câmera é inicializada com o objeto de `SurfaceHolder` recebido.

```
public CameraView(final Context context, final FonteCamera fonteCamera) {
        super(context);

        this.fonteCamera = fonteCamera;

        this.setBackgroundColor(Color.TRANSPARENT);

        final SurfaceHolder holder = this.getHolder();
        holder.addCallback(this);

        fonteCamera.init(holder);
}
```

Quadro 23 - Construtor de `CameraView`

Os métodos do ciclo de vida da *view* da câmera são todos delegados para a fonte de câmera que deve fazer as devidas tratativas de iniciar o dispositivo, ajustar o tamanho da imagem, fechar a conexão com a câmera e disponibilizar as imagens no objeto de `SurfaceHolder`, passado ao inicializar.

A segunda camada da tela é a camada da OpenGL que desenha os objetos virtuais, causando a impressão de aumento de realidade. Essa camada é implementada pela classe `RASurfaceView` e renderizada pela classe `RASurfaceRenderer`.

A classe `RASurfaceView` possui baixa complexidade pois é uma extensão da classe `GLSurfaceView` da OpenGL. A responsabilidade da classe `RASurfaceView` está em configurar a `view` para ser translúcida, instanciar e configurar a renderização. O Quadro 24 mostra a implementação do seu construtor, aonde é feita toda a configuração.

```
// Observador de mudanças na engine.
private final EngineHandler engineHandler = new EngineHandler();

public RASurfaceView(final Context context, final RAEngine raEngine) {
	super(context);

	final RASurfaceRenderer renderer = new RASurfaceRenderer(context, raEngine);

	this.setEGLConfigChooser(8, 8, 8, 8, 16, 0); // Translúcido.
	this.setRenderer(renderer); // Renderizador.
	this.setOnTouchListener(renderer); // Touch nos POIs.

	// Configura a estratégia de renderização.
	this.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);

	this.getHolder().setFormat(PixelFormat.TRANSLUCENT); // Translúcido.

	// Para funcionamento da String.
	this.setGLWrapper(GLSurfaceViewWrapper.instance);

	// Observa mudanças na engine.
	raEngine.addListener(this.engineHandler);
}
```

Quadro 24 - Configuração da OpenGL

No construtor da classe foi configurado que a estratégia de renderização é `GLSurfaceView.RENDERMODE_WHEN_DIRTY`, fazendo com que a OpenGL somente desenhe os objetos quando acionado através do método `requestRenderer`. Esse método é chamado pela classe interna `EngineHandler` que age como um observador das mudanças na *engine*. Sempre que alguma atualização ocorre na *engine*, a `EngineHandler` é avisada através do atributo `Constantes.EngineHandler_Atualiza` e chama o método `requestRenderer` para que os objetos da OpenGL sejam desenhados.

A classe responsável por mover, rotacionar e desenhar os objetos na OpenGL é `RASurfaceRender`. Essa classe implementa a interface `Renderer`, definida dentro da classe `GLSurfaceView`. O método `onSurfaceCreated`, demonstrado no Quadro 25, inicializa a OpenGL, habilitando o uso de textura em 2D, ajustando a cor de fundo para transparente, habilitando o `depth test` e configurando o `sprite` que será usado para desenhar os textos.

```
public void onSurfaceCreated(final GL10 gl, final EGLConfig config) {
	gl.glDisable(GL10.GL_DITHER);
	gl.glEnable(GL10.GL_TEXTURE_2D);
	gl.glShadeModel(GL10.GL_SMOOTH);
	gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // Transparência
	gl.glClearDepthf(1.0f);
	gl.glEnable(GL10.GL_DEPTH_TEST);
	gl.glDepthFunc(GL10.GL_LEQUAL);
	gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);

	// Inicializa o sprite do texto.
	final Paint textoPaint = new Paint();
	textoPaint.setTextSize(12);
	textoPaint.setColor(Color.BLACK);
	if (this.textoSprite != null)
		this.textoSprite.shutdown(gl);
	else
		this.textoSprite = new ASCIISprite();

	this.textoSprite.initialize(gl, textoPaint);
}
```

Quadro 25 - Inicialização da OpenGL

O método `onSurfaceChanged` é chamado quando houver alguma mudança no tamanho da tela, isso pode acontecer quando a orientação da tela é modificada de *portait* para *landscape* ou vice-versa. Nessa aplicação a orientação foi configurada (na classe `RAActivity`) para *landscape* portanto não haverá mais do que uma chamada a esse método. No Quadro 26 está sendo demonstrada a implementação desse método, que configura a `viewport`, a matriz de projeção, o `frustrum` e a matriz do modelo. Sempre que mexer na `viewport` e nas matrizes é necessário atualizar o `projector`, classe responsável por guardar as atualizações na OpenGL para realizar as mesmas transformações no texto quando for desenhado.

```
public void onSurfaceChanged(final GL10 gl, final int width, int height) {
        gl.glViewport(0, 0, width, height);
        this.projector.setCurrentView(0, 0, width, height);

        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();

        final float ratio = (float) width / height;
        gl.glFrustumf(-ratio, ratio, -1, 1, 1/*near*/, 20/*far*/);
        this.projector.getCurrentProjection(gl);

        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        this.projector.getCurrentModelView(gl);

        this.width = width;
        this.height = height;
}
```

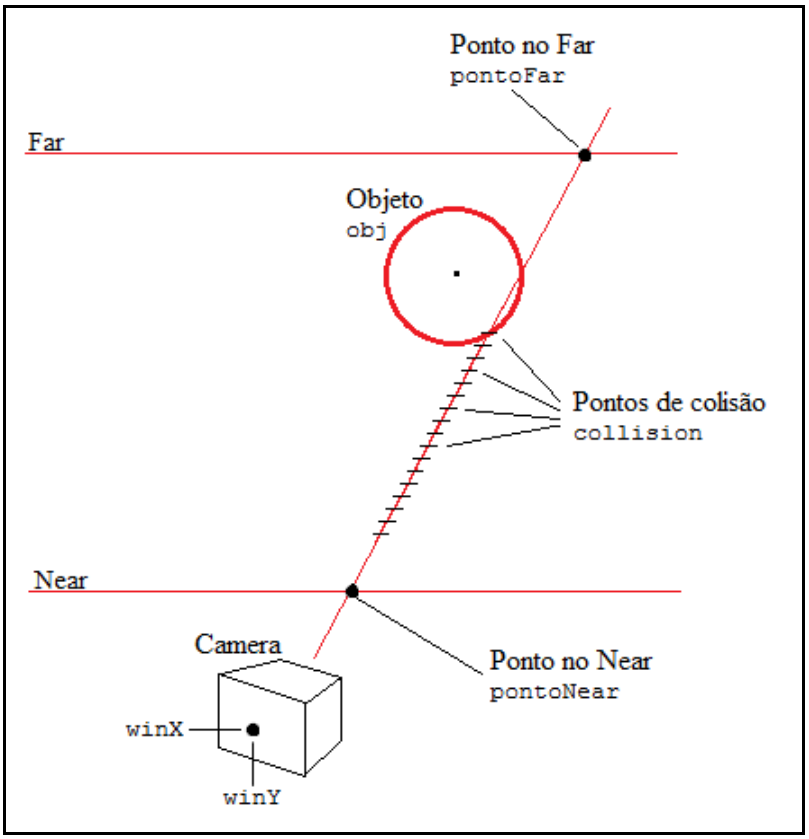
Quadro 26 - Configuração da OpenGL

O método que desenha os painéis e setas de cada ponto de interesse chama-se `onDrawFrame` e sua implementação está demonstrada no Quadro 27. Logo ao iniciar é feita uma limpeza no *buffer* de cores e no *buffer* de profundidade. Através dos valores de `roll` e `azimuth` calculados na *engine*, o `gl` é rotacionado nos eixos X e Z. Para cada ponto de interesse obtido da *engine*, desenha-se a seta que aponta para o ponto de interesse e desenha-se o painel na direção deste mesmo ponto de interesse. As classes `SetaDraw` e `PainelDraw` são utilizadas nesse método para auxiliar o desenho de cada um dos objetos virtuais.

```
public void onDrawFrame(final GL10 gl) {
        // (...)
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glLoadIdentity();

        // Rotaciona conforme a orientação
        gl.glRotatef(this.engine.roll, 1.0f, 0.0f, 0.0f);
        gl.glRotatef(this.engine.azimuth, 0.0f, 0.0f, 1.0f);
        this.projector.getCurrentModelView(gl);

        // Obtém os pontos de interesse através da engine.
        this.pois = this.engine.pois;

        // Coloca as setas que apontam para os pontos de interesse.
        final int poisLength = this.pois.length;
        for (int index = 0; index < poisLength; index++) {
                final PontoInteresse poi = this.pois[index];
                if (poi.foraTela)
                        continue;

                // Desenha a seta.
                gl.glPushMatrix();
                gl.glTranslatef(poi.glSetaX, poi.glSetaY, poi.glSetaZ);
                gl.glRotatef(-poi.glAngulo, 0.0f, 0.0f, 1.0f);
                this.setaDraw.draw(gl, (int) Math.abs(poi.geoDistancia), this.width,
this.height, this.textoSprite);
                gl.glPopMatrix();

                // Desenha o painel.
                gl.glPushMatrix();
                gl.glTranslatef(poi.glPainelX, poi.glPainelY, poi.glPainelZ);
                gl.glRotatef(-poi.glAngulo, 0.0f, 0.0f, 1.0f);
                this.painelDraw.draw(gl, poi.nome, this.width, this.height, this.textoSprite);
                gl.glPopMatrix();
        }
        // (...)
}
```

Quadro 27 - Desenhando os objetos virtuais na OpenGL

O método `onTouch` foi sobrescrito da classe base `View` para tratar o toque nas setas e nos painéis e mostrar uma janela de informações sobre o ponto de interesse tocado. A Figura 26 mostra uma representação do algoritmo utilizado para o tratamento do toque, ou também chamado `ray picking`. O toque na tela é obtido em coordenadas 2D nas variáveis `winX` e `winY`, estas devem ser convertidas para as coordenadas 3D da OpenGL. Tal conversão faz com que o ponto no plano 2D vire uma reta no espaço 3D. A reta é obtida através da conversão de `winX` e `winY` para o `near` e da mesma forma para o `far`. O algoritmo então percorre a reta que vai de `near` até `far` criando pontos de colisão, representados pela variável `collision`. Para cada ponto de colisão gerado, verifica-se se há colisão com cada ponto de interesse, representados na Figura 26 pelo `obj`.



Fonte: adaptado de Novabox (2010).

Figura 26 - Algoritmo de colisão do toque

O Quadro 28 mostra a implementação do método `onTouch` de acordo com o algoritmo `ray picking`. A conversão do ponto no plano 2D para o espaço 3D é feita pelo método `gluUnProject` implementando na mesma classe. A biblioteca OpenGL possui uma implementação padrão para `gluUnProject`, porém não foi possível utilizá-la por apresentar muitos problemas. Na referência ANDROID DEVELOPERS (2010a) há o relato de desenvolvedores que também não conseguiram utilizar a implementação padrão do Android. Na mesma referência há uma alternativa de implementação que foi incorporada a este trabalho.

```
public boolean onTouch(final View v, final MotionEvent event) {
	if (event.getAction() != MotionEvent.ACTION_DOWN) {
		return false;
	}

	// Obtém as matrizes
	this.projector.getViewport(this.viewport);
	final float[] modelView = this.projector.grabber.modelView;
	final float[] projection = this.projector.grabber.projection;

	// Obtém a posição do toque.
	final float winX = event.getX();
	final float winY = this.viewport[3] - event.getY();

	// Obtém a posição do ponto no near.
	final float[] pontoNear = new float[3];
	RASurfaceRenderer.gluUnProject(winX, winY, 0, modelView, 0, projection, 0,
this.viewport, 0, pontoNear, 0);

	// Obtém a posição do ponto no far.
	final float[] pontoFar = new float[3];
	RASurfaceRenderer.gluUnProject(winX, winY, 1, modelView, 0, projection, 0,
this.viewport, 0, pontoFar, 0);

	// Cria e normaliza o vetor ray
	// (...)

	// Percorre o vetor ray pra verificar colisões
	final float[] obj = new float[3];
	for (int iteration = 0; iteration < RAY_ITERATIONS; iteration++) {
		final float[] collision = new float[3];
		collision[0] = rayVector[0]*rayLength/RAY_ITERATIONS*iteration;
		collision[1] = rayVector[1]*rayLength/RAY_ITERATIONS*iteration;
		collision[2] = rayVector[2]*rayLength/RAY_ITERATIONS*iteration;

		// Percorre cada ponto de interesse e verifica colisão.
		final int poisLength = this.pois.length;
		for (int index = 0; index < poisLength; index++) {
			final PontoInteresse poi = this.pois[index];
			if (poi.foraTela) {
				continue;
			}

			// Verifica a colisão na seta.
			obj[0] = poi.glSetaX;
			obj[1] = poi.glSetaY;
			obj[2] = poi.glSetaZ;
			if (this.verificaColisao(obj, collision, 1.0f)) {
				this.openPOIInfo(poi);
				return true;
			}

			// Verifica a colisão no painel.
			// (...)
		}
	}
	return false;
}
```

Quadro 28 - Tratamento de colisões no toque

As classes `PainelDraw` e `SetaDraw` desenham de maneira semelhante as formas geométricas, preparando os vetores e os *buffers* nos seus construtores e desenhando-os utilizando as primitivas no método `draw`, conforme demonstrado no Quadro 29. Dentre as três funcionalidades demonstradas na sessão 2.2.5 para desenho de objetos na OpenGL, a utilizada foi a de vértices básicos por ser suportada por todos os dispositivos da plataforma Android.

A última camada da tela mostra informações e permite fazer configurações que auxiliam a segunda camada, ela possui o desenho da bússola, do alcance da tela, do serviço de

localização e do medidor de desempenho. A classe principal desta última camada é a `InfoView` e sua responsabilidade é direcionar o desenho e o toque para as demais classes.

```
public void draw(/*Parâmetros omitidos*/) {
	gl.glFrontFace(GL10.GL_CW);

	gl.glColor4f(1.0f, 0.0f, 0.0f, 0.0f); // Vermelho
	gl.glVertexPointer(3, GL10.GL_FLOAT, 0, this.buffer);
	gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
	gl.glDrawArrays(GL10.GL_TRIANGLES, 0, this.qtdeTriangulos);
	gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);

	// (...)
}
```

Quadro 29 - Desenho de um painel

Dentre as classes que se destacam nesta camada está a classe `RadarView` que desenha um radar com os pontos de interesse e com os pontos cardeais. O Quadro 30 mostra o principal método dessa classe, responsável por desenhar todos os elementos do radar.

```
protected void onDraw(final Canvas canvas) {
	final float raio = this.preferencias.getRadarTelaRaio();

	canvas.save();

	// Translada para o ponto central do radar.
	canvas.translate(this.x, this.y);

	// Pinta o círculo do radar
	canvas.drawCircle(0.0f, 0.0f, raio, this.circuloPaint);

	// Rotaciona os textos e os pois.
	canvas.rotate(-this.engine.azimuth);

	// Pinta os pontos de interesse no radar.
	final PontoInteresse[] pois = this.engine.pois;
	final int poisLength = pois.length;
	for (int indice = 0; indice < poisLength; indice++) {
		final PontoInteresse poi = pois[indice];
		if (!poi.foraRadar) {
			canvas.drawCircle(poi.glRadarX, -poi.glRadarY, 3, this.textoPaint);
		}
	}

	final Rect bounds = new Rect();

	// Pinta o texto do Norte.
	final String n = "N";
	this.paint.getTextBounds(n, 0, n.length(), bounds);
	canvas.drawText(n, -bounds.right / 2, -raio, this.paint);

	// Pinta o texto do Sul, Leste e Oeste.
	// (...)

	canvas.restore();
}
```

Quadro 30 - Desenho do radar

3.4.5 Desenho de texto na OpenGL

As classes da aplicação que fornecem a funcionalidade de desenho de texto na OpenGL foram obtidas através da referência Google (2010m) e adaptadas à necessidade de

desenho de textos dinâmicos que a API original não possui.

A classe `LabelMaker` gerencia os textos dentro de um objeto de `Canvas` e projeta para um objeto de `GL10` através do uso de texturas para cada texto. Todos os seus métodos tiveram seu escopo alterado para `package` para fortalecer o encapsulamento dos recursos disponibilizados por essa classe. Seus três principais métodos `beginDrawing`, `draw` e `endDrawing` realizam o desenho das texturas previamente carregadas, conforme o Quadro 31. O método `beginDrawing` prepara a textura para ser desenhada e deve ser chamado apenas uma vez. O método `draw` realiza a pintura do texto cujo identificador foi passado no parâmetro `labelId`, devendo ser chamado inúmeras vezes. Por último o método `endDrawing` desabilita as funções e desempilha os estados que foram habilitados no `beginDrawing`, devendo ser chamado também apenas uma vez antes da próxima chamada à `beginDrawing`.

```
void beginDrawing(final GL10 gl, final float width, final float height) {
        // (...)
        gl.glBindTexture(GL10.GL_TEXTURE_2D, this.textureID);
        gl.glShadeModel(GL10.GL_FLAT);
        gl.glEnable(GL10.GL_BLEND);
        gl.glBlendFunc(GL10.GL_SRC_ALPHA, GL10.GL_ONE_MINUS_SRC_ALPHA);
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // (...)
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        // (...)
}

void draw(final GL10 gl, final float x, final float y, final int labelID){
        // (...)
        final Label label = this.labels.get(labelID);
        gl.glEnable(GL10.GL_TEXTURE_2D);
        ((GL11) gl).glTexParameteriv(GL10.GL_TEXTURE_2D, GL11Ext.GL_TEXTURE_CROP_RECT_OES,
label.crop, 0);
        ((GL11Ext) gl).glDrawTexiOES((int) x, (int) y, 0, (int) label.width, (int)
label.height);
        gl.glDisable(GL10.GL_TEXTURE_2D);
}

void endDrawing(final GL10 gl) {
        gl.glDisable(GL10.GL_BLEND);
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glPopMatrix();
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glPopMatrix();
}
```

Quadro 31 - Desenho dos textos

A classe `ASCIISprite` foi criada nessa aplicação para utilizar a classe `LabelMaker` e criar um `sprite` para cada caractere da tabela ASCII. Como a classe `LabelMaker` trabalha de forma que existe um determinado momento para poder adicionar os textos em texturas e após esse tempo não poderá haver nenhuma mudança, a classe `ASCIISprite` utiliza-se desse momento para adicionar todos os caracteres da tabela ASCII. No momento do desenho os caracteres serão desenhados um ao lado do outro para montar as palavras desejadas. O método `initialize` faz tal preparação e sua implementação está sendo mostrada no Quadro 32.

```
public void initialize(final GL10 gl, final Paint paint) {
        // (...)
        // Agrupa os espaços entre os dígitos.
        final float interDigitGaps = (QtdeCodigosAscii - 1) * 1.0f;

        // (...)
        // Monta a classe que gerencia as texturas de labels.
        this.labelMaker = new LabelMaker(width, height);
        this.labelMaker.initialize(gl);

        // Adiciona um por um todos os caracteres.
        this.labelMaker.beginAdding();
        for (int codigoASCII = 0; codigoASCII < QtdeCodigosAscii; codigoASCII++) {
                final String caracterASCII = String.valueOf(CaracteresAscii[codigoASCII]);
                final int idTextura = this.labelMaker.add(caracterASCII, paint);
                this.labelIDs[codigoASCII] = idTextura;
                this.widths[codigoASCII] = (int)
Math.ceil(this.labelMaker.getWidth(idTextura));
                this.heights[codigoASCII] = (int)
Math.ceil(this.labelMaker.getHeight(idTextura));
        }
        this.labelMaker.endAdding(gl);
}
```

Quadro 32 - Inicialização dos *sprites* ASCII

O método de desenho da classe `ASCIISprite` percorre cada caractere e delega para a classe `LabelMaker` desenhá-los um ao lado do outro, conforme pode ser conferido no Quadro 33.

```
public void draw(final GL10 gl, final String text, float x, final float y, final float width,
final float height) {
        final int length = text.length();

        // Pinta cada caractere do texto.
        this.labelMaker.beginDrawing(gl, width, height);
        for (int index = 0; index < length; index++) {
                final int codigoAscii = text.charAt(index);
                this.labelMaker.draw(gl, x, y, this.labelIDs[codigoAscii]);
                x += this.widths[codigoAscii];
        }
        this.labelMaker.endDrawing(gl);
}
```

Quadro 33 - Desenho dos caracteres

3.4.6 Simulação da câmera

A interface `FonteCamera` define uma série de métodos para tratar as imagens vindas de uma fonte de câmera. Através dela foi possível definir uma implementação de câmera para o simulador. A classe `SimulacaoCamera` possui toda a complexidade de obtenção das imagens da câmera através de *socket* com um programa que roda no computador do simulador.

O programa que disponibiliza as imagens da câmera através de *socket* foi obtido através na referência Tom Gibara (2010). Sua implementação utiliza a API do Java *Media Framework* (JMF) para obter as câmeras conectadas ao computador e tratar o *buffer* de imagens.

A classe `SimulacaoCamera` possui uma *thread* interna que busca as imagens da câmera e coloca na tela através do objeto da classe `SurfaceHolder`. A conexão de *socket* possui um tempo máximo de espera para evitar que a *thread* fique aguardando uma conexão inexistente. O Quadro 34 mostra a execução da *thread* para obter e disponibilizar as imagens da câmera. Em primeiro momento a tela é travada para modificações através da chamada ao método `holder.lockCanvas`, no final da execução a tela é destravada e atualizada através da chamada ao método `holder.unlockCanvasAndPost`. Faz-se uma comunicação via *socket* através da classe `Socket` e por ela é obtida a imagem da câmera no formato de um `InputStream`. Através da classe `BitmapFactory` foi possível converter o objeto de `InputStream` em um mapa de bits, representado pela classe `Bitmap`.

```
public void run() {
	while (this.running) {

		// Bloqueia parte do canvas para pintar.
		final Canvas canvas = this.holder.lockCanvas();

		Socket socket = new Socket();
		socket.bind(null);
		socket.setSoTimeout(SimulacaoCamera.SocketTimeout);
		socket.connect(new InetSocketAddress(Constantes.SocketEnderecoIP,
Constantes.SocketCameraPorta), SimulacaoCamera.SocketTimeout);

		// Obtém o bitmap
		final InputStream in = socket.getInputStream();
		final Bitmap bitmap = BitmapFactory.decodeStream(in);
		if (bitmap == null) {
			return;
		}

		// Renderiza no canvas.
		// (...)
		canvas.drawBitmap(bitmap, 0, 0, null);
		// (...)

		// Desbloqueia o canvas e atualiza a pintura.
		this.holder.unlockCanvasAndPost(canvas);
		// (...)
	}
}
```

Quadro 34 - Obtendo as imagens da câmera por *socket*

3.4.7 Simulação dos sensores

Uma fonte de sensor é representada genericamente pela classe `FonteSensor`, sendo sua responsabilidade informar à aplicação que houve alguma mudança nos sensores, quais foram eles e quais os dados mais atuais. A classe `SimulacaoSensor` é a responsável por fornecer dados simulados de sensores através de *socket* comunicando-se com um programa que roda no computador do simulador.

Para fornecer os dados simulados dos sensores foi encontrado o programa *Sensor*

Simulator (OPEN INTENTS, 2010a) que possui um programa servidor de *socket* alimentado por uma tela e um aplicativo cliente para rodar no dispositivo Android. O programa servidor pôde ser aproveitado por completo para a aplicação deste trabalho. Já o aplicativo cliente precisou ser bastante adaptado, pois a versão do Android no qual ele está compatível é a 1.1 e este trabalho foi desenvolvido na versão 2.2. O aplicativo cliente possui ainda, classes para gerenciamento de telas, banco de dados e comunicação com o servidor *socket*. Destas funcionalidades foi aproveitada apenas a classe `SensorSimulatorClient`, disponível na referência *Open Intents* (2010b), que se comunica com o servidor e sua implementação foi incorporada pela classe `SimulacaoSensor`.

Na classe `SimulacaoSensor` há uma *thread* que busca os valores de cada sensor utilizando o método `socketLerSensor`. Tal método envia para o servidor *socket* o comando `readSensor()` seguido do nome do sensor, o servidor envia várias mensagens de retorno. O primeiro retorno é a quantidade de valores a serem enviados e os retornos seguintes são os valores. Os retornos do servidor são obtidos através da chamada ao método `entrada.readLine()` conforme demonstrado no Quadro 35.

```
private float[] socketLerSensor(final String sensor) {
        this.saida.println("readSensor()");
        this.saida.println(sensor);
        // (...)
        final String linha = this.entrada.readLine();
        // (...)
        final int qtdeValores = Integer.parseInt(linha);
        final float[] valores = new float[qtdeValores];
        for (int indice = 0; indice < qtdeValores; indice++) {
                final String valor = this.entrada.readLine();
                valores[indice] = Float.parseFloat(valor);
        }
        return valores;
        // (...)
}
```

Quadro 35 - Leitura de dados de um sensor simulado

3.4.8 Medição do desempenho da aplicação

O mesmo recurso utilizado por Google (2010l) para medir o desempenho da OpenGL em comparação com o do `Canvas`, descrito na sessão 2.2.5, foi utilizado neste projeto para medir o desempenho da aplicação. Foi aproveitada a classe `ProfileRecorder`, precisando apenas adaptar os tipos numéricos para `float` para não perder a precisão nas operações de divisão. A medição está sendo feita nos dois principais processos aplicação: os cálculos da *engine* (na classe `RAEngine`) e o desenho dos objetos na OpenGL (na classe `RASurfaceRenderer`). O Quadro 36 mostra o medidor de desempenho sendo utilizado na

classe `RASurfaceRenderer`.

```
public void onDrawFrame(final GL10 gl) {

        // Inicia o teste de desempenho.
        this.engine.profile.start(ProfileRecorder.PROFILE_DRAW);

        // (...)

        // Finaliza o teste de performance.
        this.engine.profile.stop(ProfileRecorder.PROFILE_DRAW);
        this.engine.profile.endFrame();
}
```

Quadro 36 - Medição do desempenho da OpenGL

3.4.9 Servidor web de pontos de interesse

O servidor *web* que fornece os pontos de interesse possui a classe `FontePOIsServlet` que é uma implementação de um `Servlet` definido pela arquitetura `JEE`. Seu método principal, `doGet` (Quadro 37), conecta-se com um banco de dados através da interface JDBC e obtém o resultado de um `Select` na tabela de pontos de interesse. O resultado do `Select` é adicionado à resposta do `Servlet` no objeto da classe `HttpServletResponse`.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        // Obtém uma conexão com o banco.
        final Connection conn = DriverManager.getConnection(ODBC_CONNECTION, ODBC_USER,
ODBC_PASS);

        // (...)
        StringBuilder out = new StringBuilder();
        Statement stmt = conn.createStatement();
        ResultSet result = stmt.executeQuery("SELECT nome,latitude,longitude FROM
PontosInteresse");

        // (...)
        while (result.next()) {
                String nome = result.getString("nome");
                String latitude = result.getString("latitude");
                String longitude = result.getString("longitude");

                out.append(nome);
                out.append(VALUE_SEPARATOR);
                out.append(latitude);
                out.append(VALUE_SEPARATOR);
                out.append(longitude);
                out.append(LINE_SEPARATOR);
        }
        // (...)
        // Escreve os pontos de interesse na saída.
        ServletOutputStream output = response.getOutputStream();
        output.print(out.toString());
        output.flush();
        // (...)
}
```

Quadro 37 - Obtenção dos pontos de interesse no banco de dados

3.4.10 Operacionalidade da aplicação

A operacionalidade da aplicação é apresentada na forma de funcionalidades e casos de uso, sendo representados através de imagens. Serão apresentadas imagens tanto da aplicação funcionando no simulador quanto no dispositivo HTC Desire, ambos utilizados durante o desenvolvimento do trabalho.

A aplicação foi implementada de forma a possuir uma tela inicial de menu que permite entrar nas configurações da aplicação e também entrar na tela de realidade aumentada que possui as três camadas descritas anteriormente. A tela de realidade aumentada possui as funcionalidades que conferem à aplicação o conceito de realidade aumentada, fazendo uso de telas modais quando necessário configurar o radar e o alcance.

No momento que a aplicação é iniciada é aberta a tela de menu que possui dois botões e uma imagem, conforme pode ser conferido na Figura 27.

Figura 27 – Tela do menu da aplicação

Através do botão `Configurações` é possível entrar na configuração da aplicação (Figura 28) e definir valores para o alcance do radar, alcance da tela e URL do servidor que fornece os pontos de interesse. Quando no simulador, os IPs e portas para as fontes de sensor e de câmera podem ser configurados nessa tela também.

Figura 28 - Tela de configuração da aplicação

Quando o botão `Realidade Aumentada` é pressionado, as três camadas são desenhadas, conforme pode ser conferido na Figura 29. A imagem de fundo é proveniente da câmera do dispositivo e os componentes nas laterais são o radar (no canto direito acima), a localização (no canto direito abaixo), o medidor de desempenho (no centro abaixo) e o alcance (no canto esquerdo abaixo). Na Figura 29 não há o desenho de nenhum ponto de interesse, pois a localização geográfica ainda não foi carregada pelo dispositivo.

Figura 29 – Tela de realidade aumentada no HTC Desire

3.4.10.1 O aumento de realidade

Assim que a localização geográfica for obtida pelo dispositivo, a aplicação desenha os pontos de interesse na forma de setas, painéis e pontos no radar. Para visualizar as setas é necessário direcionar a câmera do dispositivo para o chão e a tela para cima. A Figura 30 mostra o desenho das setas e os pontos no radar apontando para a direção de cada ponto de interesse. O número dentro da seta indica a distância que o ponto de interesse se encontra.

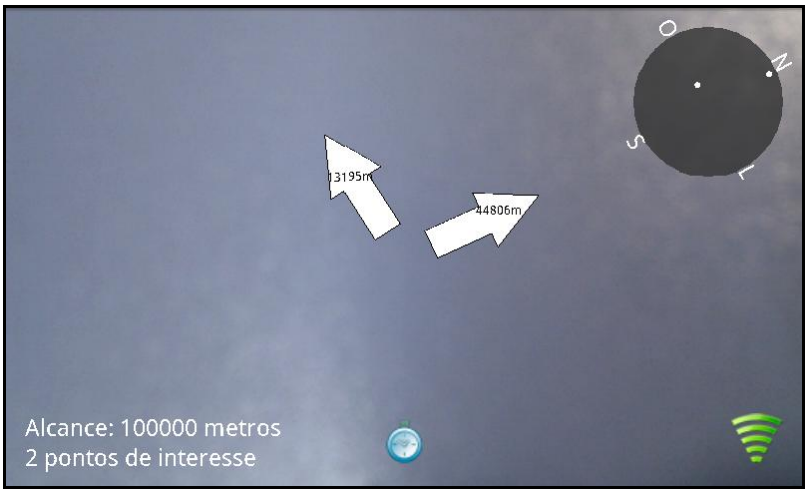


Figura 30 - Desenho das setas

Os painéis são visualizados quando a câmera é direcionada para frente e o dispositivo estiver sendo segurado na altura dos ombros, conforme mostra a Figura 31. Cada painel possui o nome do ponto de interesse que está sendo representado.

Figura 31 - Desenho dos painéis

3.4.10.2 Configurações e medições

Os quatro desenhos das laterais da tela (radar, localização, medidor de desempenho e alcance) possuem tratamento ao toque e através dele executam alguma atividade em particular. Ao tocar no desenho do radar é aberta uma tela modal para a configuração do alcance deste radar, conforme pode ser conferido na Figura 32. Por ser uma tela modal, a aplicação continua sendo desenhada abaixo dela.

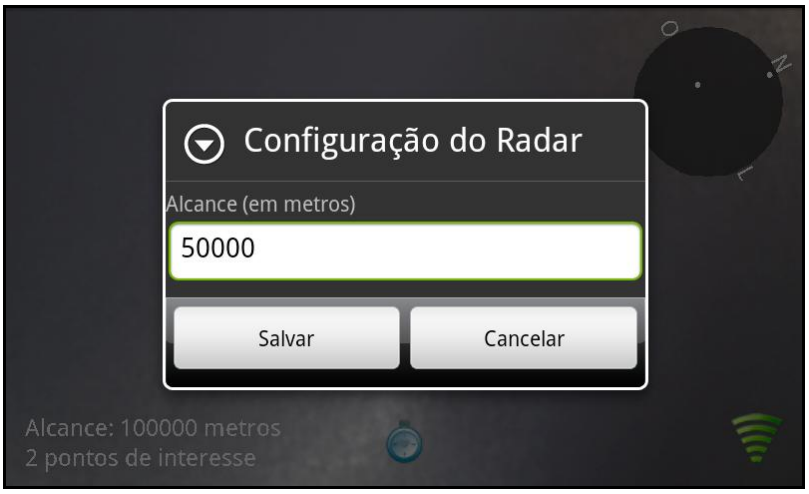


Figura 32 - Tela modal para configuração do radar

A configuração do alcance da tela também possui uma tela modal (Figura 33), sendo seu objetivo configurar o seu valor em metros. Tanto o botão `Salvar` quanto o botão `Cancelar` fecham a tela modal, sendo o primeiro também responsável por gravar as alterações feitas na tela.

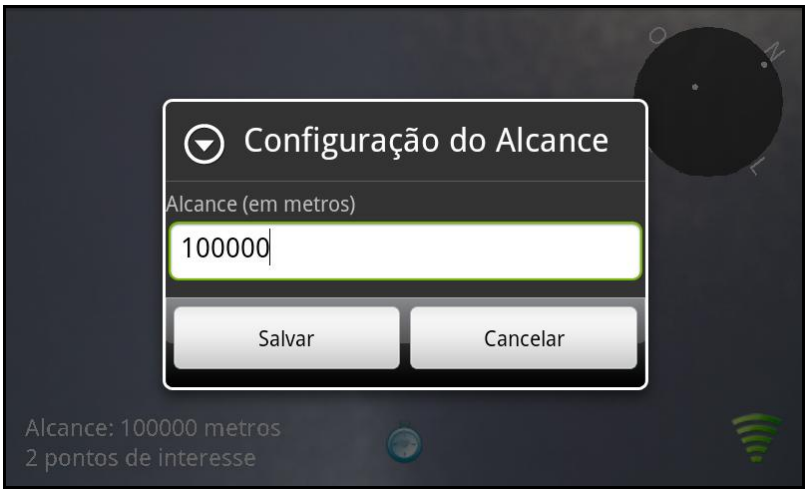


Figura 33 - Tela modal para configuração do alcance da tela

Ao tocar no ícone que está no canto inferior direito da tela, aparecem várias informações sobre a localização geográfica como latitude, longitude, altitude, fonte de localização geográfica e a sua precisão (Figura 34). Essa mensagem desaparece da tela após certo tempo de espera.

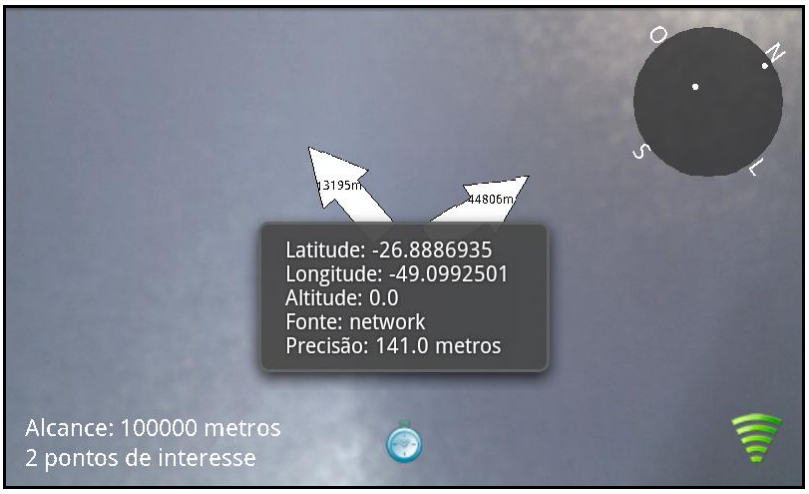


Figura 34 - Informações da localização

Da mesma forma ao tocar no ícone inferior central da tela, aparecem informações relativas ao desempenho da aplicação, medidas em FPS tanto na *engine* quanto no desenho da tela (Figura 35). Os valores informados são a média, o mínimo e o máximo FPS obtidos durante a execução da aplicação.

Figura 35 - Medição do desempenho

Ao tocar em uma seta ou em um panel de um ponto de interesse, é aberta uma tela modal (Figura 36) com informações do ponto de interesse tocado. O botão `Pesquisa` leva a uma tela de *browser* que pesquisa informações do ponto de interesse na *internet*.

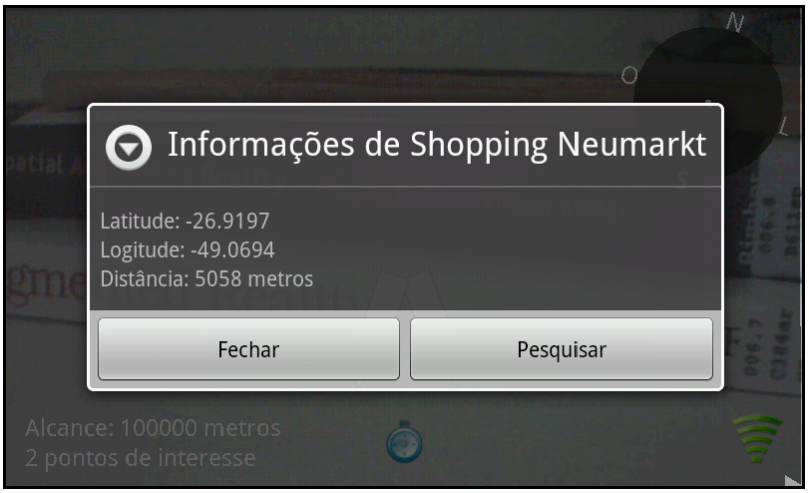


Figura 36 - Informações de um ponto de interesse a partir do toque

3.4.10.3 Funcionalidades do simulador

No simulador a aplicação consegue manter o mesmo comportamento do dispositivo, mostrando as imagens da câmera, a movimentação nos sensores e nas coordenadas geográficas. A Figura 37 mostra um dispositivo simulado que possui a mesma resolução que o HTC Desire.

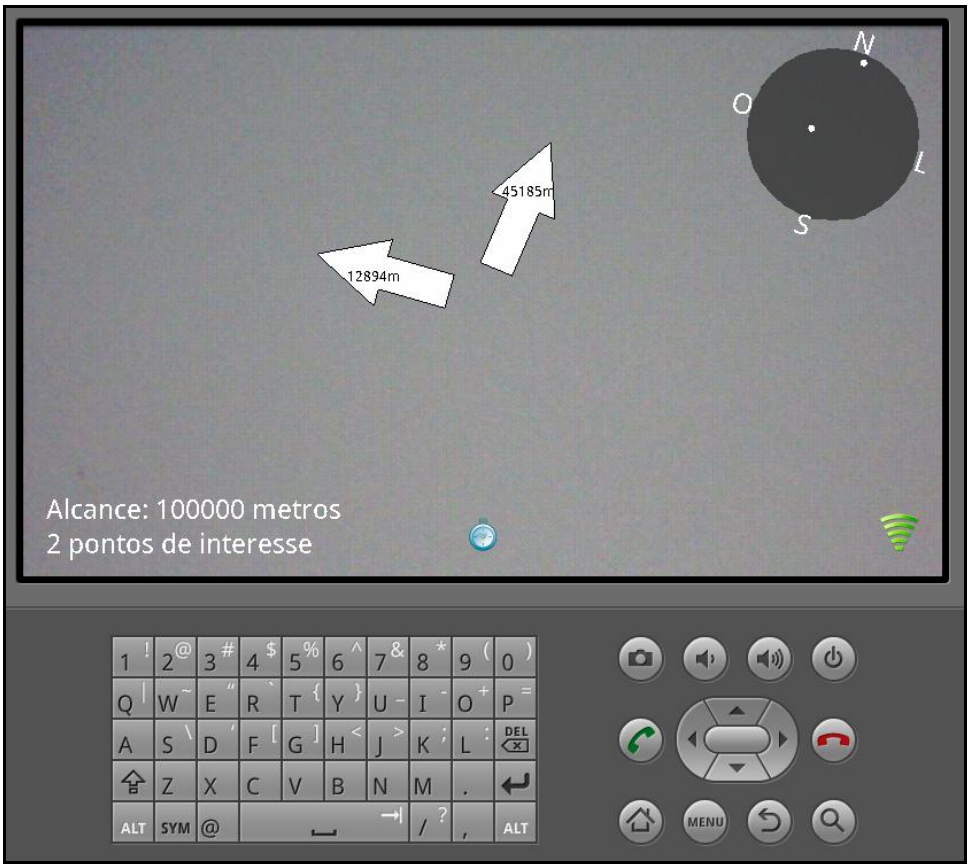


Figura 37 - Aplicação rodando no simulador

Para simular a movimentação das coordenadas geográficas, as teclas tratadas pela aplicação são `DPad left`, `DPad right`, `DPad up` e `DPad down`, todas demonstradas pela Figura 38.

Figura 38 - Botões para simular a mudança de localização

3.4.10.4 Simular os sensores

Entre os resultados obtidos neste trabalho está o conjunto de fontes para rodar o programa servidor que simula os dados dos sensores. Esse programa possui uma tela principal (Figura 39) aonde podem ser feitas todas as configurações necessárias para a simulação. Para facilitar a descrição foram criadas marcações na Figura 39 e essas serão descritas na ordem numérica. A primeira marcação, `#1`, mostra um desenho que representa um dispositivo, na medida em que a simulação acontece esse dispositivo movimenta-se facilitando a visualização da situação dele. A marcação `#2` mostra três barras do tipo *slider* que permitem alterar os valores de *yaw*, *pitch* e *roll* do -180º até o +180º. O valor de *yaw* possui o mesmo valor que o *azimuth* e portanto é interpretado como tal. A terceira marcação, `#3`, está grifando a configuração da porta na qual serão disponibilizados os valores dos sensores.

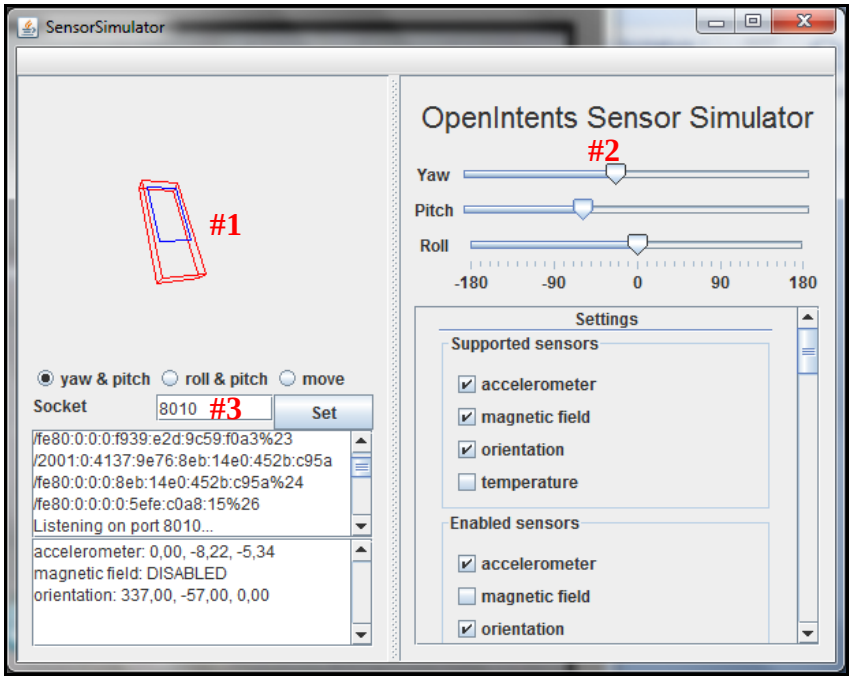


Figura 39 - Programa para simulação de sensores

3.4.10.5 Simular a câmera

Outro resultado obtido neste trabalho é o programa servidor que simula a câmera de um dispositivo. Esse programa possui apenas uma tela para seleção do dispositivo que será usado como fonte de captura das imagens, vide Figura 40. Após a seleção da câmera o programa começa a disponibilizar as imagens via *socket* para serem acessadas pela aplicação de realidade aumentada no IP do computador.

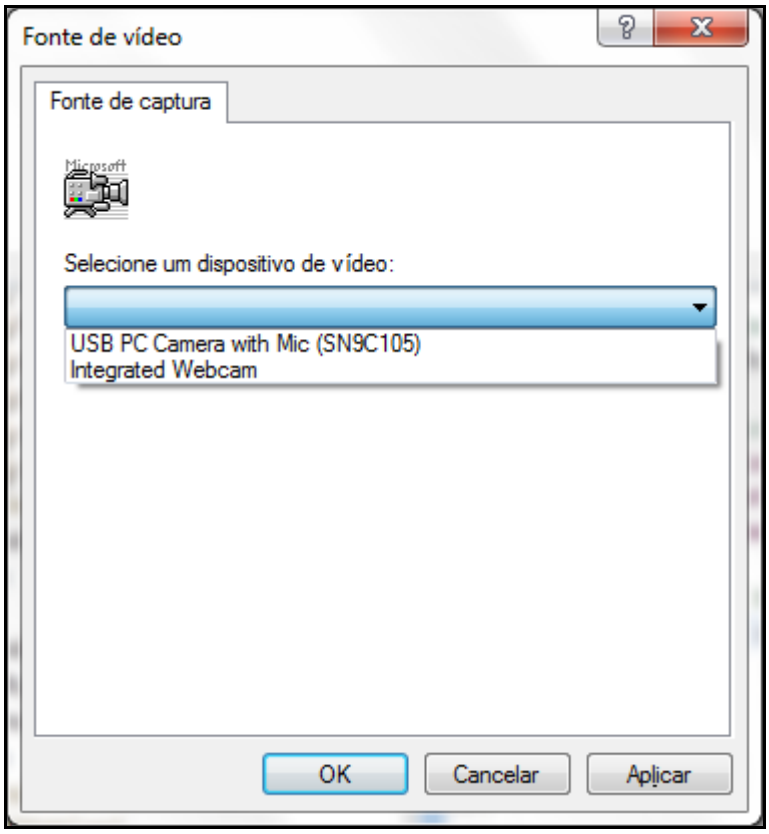


Figura 40 - Tela para a escolha da câmera para simulação

3.5 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o projeto e o desenvolvimento de um aplicativo que segue o conceito de realidade aumentada definido por Azuma (1997, p. 2) e que possui as três propriedades básicas para ser considerado como tal, conforme explicado na sessão 2.1. A primeira propriedade trata da inter-relação entre a realidade e a virtualidade na utilização de algum dispositivo, sendo atendida nesse trabalho pela utilização da própria plataforma

Android que possibilitou o uso de imagens reais e o desenho de imagens virtuais. A segunda propriedade trata do registro e rastreamento dos objetos virtuais no mundo real e foi atendida pelo registro dos objetos através das suas coordenadas geográficas. A última propriedade trata da interatividade dos objetos virtuais, que pôde ser atendida através da utilização dos sensores de acelerômetro e bússola, permitindo aplicar a movimentação do mundo real nos objetos virtuais.

Este trabalho também teve por desafio tornar viável o desenvolvimento de aplicativos com realidade aumentada dentro do simulador do Android. Não fazendo parte dos objetivos iniciais do trabalho, esse desafio pode ser considerado como uma dificuldade encontrada durante o desenvolvimento do trabalho. Também a falta de conhecimento em bibliotecas que façam uso de dispositivos de câmera, como o JMF, teve grande impacto nesse desafio.

Outra grande dificuldade encontrada foi a de desenhar texto através da biblioteca da OpenGL ES. Sua especificação não possui suporte a esse recurso e boa parte das fontes de pesquisa encontradas mostravam soluções vagas e abstratas. A solução encontrada ainda possui grandes limitações e problemas que devem ser destacados. O primeiro deles é a falta de suporte do texto ao *depth test*, fazendo com que os textos sejam desenhados e sobrepostos sem respeitar colisões ou ordem das camadas do desenho, portanto dificultando a leitura. Outro problema encontrado refere-se à localização do texto na OpenGL, calculado na classe `Projector` que em determinados momentos posiciona o texto também no extremo oposto do espaço aonde ele deveria estar.

A sobreposição das três camadas de tela também foi um desafio neste trabalho. A ordem de adição das camadas não obedece à ordem lógica de visualização delas, sendo adicionada primeiro a camada da OpenGL como camada principal, seguida da camada da câmera e por fim pela camada das ferramentas. Soma-se a este desafio a configuração da OpenGL e da camada de câmera para serem translúcidas, demandando longas horas de pesquisa e testes repetitivos.

Outra dificuldade encontrada foi na utilização da API da OpenGL ES 1.0 como um todo, suas limitações, a escassez de documentação e os *bugs*. Foi encontrado um problema no desenho da camada da OpenGL que limita o seu uso na sobreposição de camadas. Ao sair da aplicação com o botão `home` e abri-la novamente, a camada da OpenGL passa a ser desenhada, pelo gerenciador de janelas do Android, abaixo da camada de câmera fazendo com que ela não seja mais visível. A referência ANDROID DEVELOPERS (2010b) documenta que o gerenciador de janelas do Android não garante qual ordem será utilizada para desenhar as camadas sobrepostas. A referência em questão diz ainda que para contornar o problema pode-

se utilizar o recurso de `z-order`, porém testes efetuados na aplicação mostraram que esse recurso não resolve o problema. Como solução de contorno ficou estabelecido que a tela de realidade aumentada nunca entra no estado `Pausado`, assim que o Android tenta pausar a aplicação é chamado o método `finish` para destruí-la.

A última dificuldade identificada diz respeito ao simulador e ao dispositivo. O simulador do Android apresentou uma quantidade grande de limitações para desenvolvimento de aplicativos de realidade aumentada, motivo pelo qual foram implementados neste trabalho principalmente os recursos de simulação da câmera e simulação dos sensores. Além das limitações, o baixo desempenho do simulador apresentou uma dificuldade que só pôde ser contornada pelo uso de um dispositivo. A disponibilidade de dispositivos no Brasil que permitam o desenvolvimento e a depuração foi outro fator que atrapalhou o desenvolvimento. O dispositivo HTC Desire utilizado no desenvolvimento deste trabalho ainda não foi lançado no Brasil e o proprietário do aparelho o disponibilizou apenas nos finais de semana. O próprio dispositivo apresenta um problema com a bússola interna que por vezes fica descalibrada, forçando o usuário a fazer um movimento em forma de oito para voltar a ficar calibrado. Numa pesquisa breve obteve-se a informação de que, em aparelhos de outras marcas que possuem bússola, também acontece a mesma descalibragem.

3.5.1 Resultados obtidos nos testes de desempenho

O método utilizado para medir o desempenho da OpenGL em comparação com o Canvas, descrito na sessão 2.2.5, demonstrou a superioridade da OpenGL em desenhar grandes quantidades de objetos. Esse teste não só ajudou a decidir qual das duas tecnologias de tela seria adotada para esse trabalho como também forneceu as ferramentas para medição de desempenho prático que foram utilizadas nesse trabalho.

As medições do FPS foram obtidas em dois trechos de código da aplicação. O primeiro trata-se do método `atualizaPOIs` da classe `RAEngine` que faz todos os cálculos dos pontos de interesse e o segundo trecho de código é o método `onDrawFrame` da classe `RASurfaceRenderer` que faz o desenho dos pontos de interesse.

Foram efetuados testes envolvendo diferentes quantidades de pontos de interesse a serem calculados e desenhados na tela. As quantidades foram de 2, 4, 8, 16, 32 e 64 pontos de interesse calculados, sendo que para cada ponto de interesse são desenhados dois objetos na OpenGL, a seta e o painel.

Todos os testes foram executados da mesma maneira utilizando tanto o simulador quanto o dispositivo HTC Desire. A aplicação possui códigos diferentes quando executada no dispositivo e no simulador apenas nas classes de *hardware* (câmera, sensores e coordenadas geográficas) que não são medidas nos testes, portanto não devem influenciar nas medições.

Na implementação da aplicação desse trabalho foram levadas em consideração todas as otimizações descritas na sessão 2.2.8, mas também foi implementada uma versão da aplicação sem as otimizações, para ser utilizada nos testes de desempenho prático.

Os testes de desempenho então foram executados contendo certa quantidade de objetos (2, 4, 8, 16, 32 ou 64), em um dos aparelhos (simulador ou dispositivo) tanto na aplicação que contém otimizações quanto na que não contém.

O primeiro conjunto de testes foi executado no simulador utilizando a aplicação que não contém nenhuma otimização. O resultado do FPS obtido tanto na `RAEngine` quanto na `RASurfaceRenderer` está disponível na Tabela 1. Observa-se que o valor do FPS na interface gráfica está bem abaixo do ideal.

Tabela 1 - Medições no simulador usando a aplicação sem otimizações

Quantidade de pontos de interesse	**Média FPS na *engine***	**Média FPS na interface gráfica**
2	455,09	18,52
4	194,74	11,56
8	39,68	5,32
16	9,10	3,67
32	3,45	1,72
64	1,84	0,95

O segundo conjunto de testes também utilizou o simulador, mas com a aplicação que contém as otimizações. O valor do FPS na interface gráfica e na *engine* tiveram uma leve melhora, conforme pode ser conferido na Tabela 2.

Tabela 2 - Medições no simulador usando a aplicação com otimizações

Quantidade de pontos de interesse	**Média FPS na *engine***	**Média FPS na interface gráfica**
2	502,65	22,78
4	283,74	13,03
8	91,51	4,50
16	15,69	3,75
32	5,48	2,43
64	2,61	1,11

O terceiro (Tabela 3) e o quarto (Tabela 4) conjuntos de testes foram executados no dispositivo HTC Desire e os resultados das medições foram muito superiores aos resultados obtidos no simulador.

Tabela 3 - Medições no dispositivo usando a aplicação sem otimizações

Quantidade de pontos de interesse	**Média FPS na *engine***	**Média FPS na interface gráfica**
2	1248,00	93,27
4	1112,75	54,94
8	774,93	41,07
16	587,28	27,01
32	309,25	16,27
64	123,71	11,93

Não somente houve uma melhora no desempenho da *engine* como houve uma melhora considerável no desempenho da interface gráfica. O terceiro conjunto de testes utilizou a aplicação que não contém as otimizações e pode ser conferido na Tabela 3.

A quantidade de objetos afeta bastante no desempenho tanto da *engine* quanto da interface gráfica. O desempenho da aplicação com otimizações mostrou ser bastante superior ao desempenho da aplicação sem otimizações. Os resultados da quarta medição, que utiliza a aplicação que contém otimizações pode ser conferido na Tabela 4.

Tabela 4 - Medições no dispositivo usando a aplicação com otimizações

Quantidade de pontos de interesse	**Média FPS na *engine***	**Média FPS na interface gráfica**
2	4273,17	106,25
4	1897,56	58,22
8	1498,11	49,44
16	979,24	27,96
32	792,00	17,08
64	286,22	12,32

Com base nas quatro medições da *engine* foi possível gerar o gráfico da Figura 41 que representa o desempenho obtido através da medição do FPS na vertical e da quantidade de pontos de interesse na horizontal. Quase não é possível notar a diferença da otimização quando a aplicação rodou no simulador, porém no dispositivo HTC Desire a diferença ficou bastante perceptível quando feita a medição com 2 à 32 pontos de interesse.

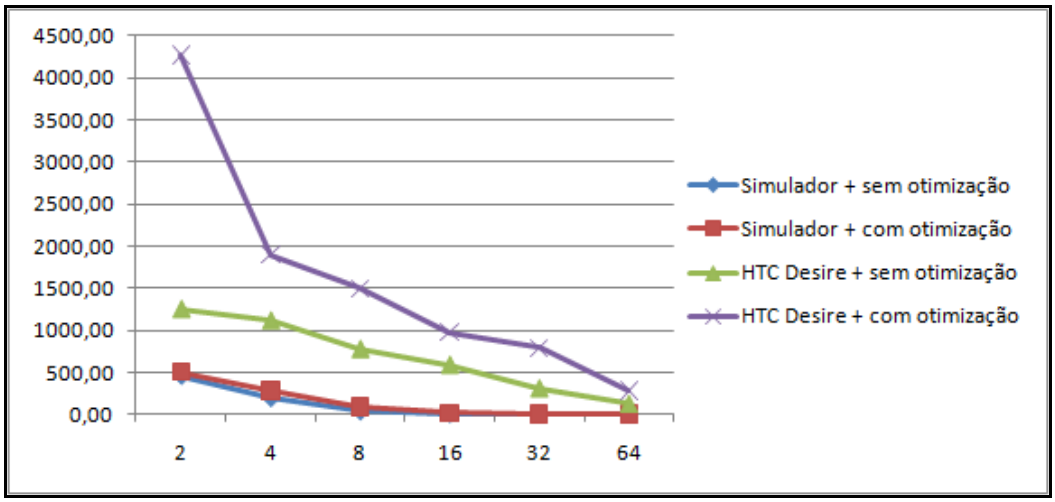


Figura 41 - Gráfico do desempenho da *engine*

Também foi possível gerar o gráfico do desempenho da interface gráfica (Figura 42) através das quatro medições. Diferente da *engine*, na interface gráfica é pouco notável o desempenho das aplicações com e sem otimização, mas é bastante perceptível a diferença no desempenho do simulador para o dispositivo HTC Desire.

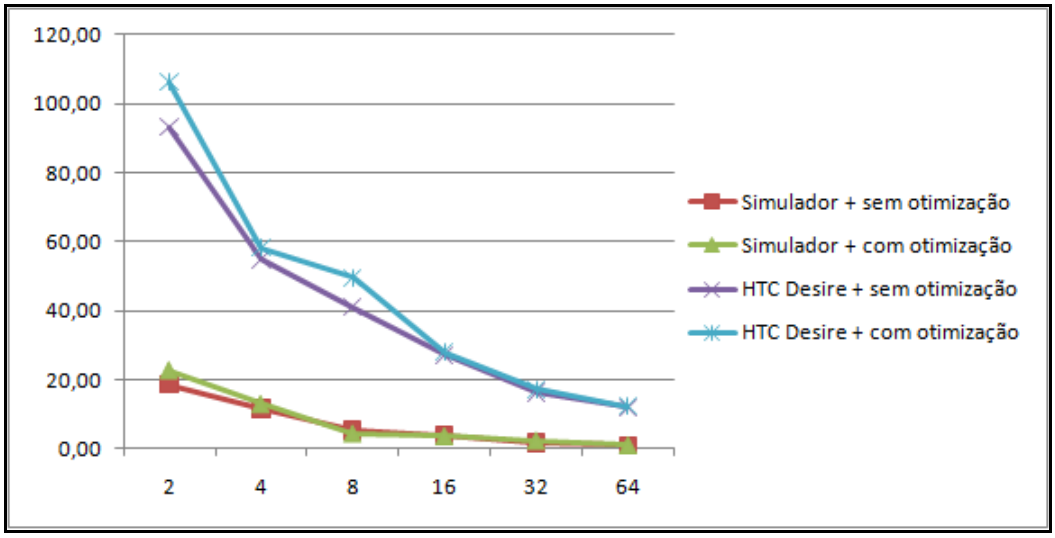


Figura 42 - Gráfico do desempenho da interface gráfica

Através desses testes foi possível perceber que as otimizações realizadas na aplicação quase não afetaram o desempenho da interface gráfica, porém na *engine* foi possível medir a importância de seguir as recomendações de otimização descritas na sessão 2.2.8.

Os resultados obtidos pelos testes de desempenho no simulador do Android estão muito abaixo dos resultados do dispositivo HTC Desire, mesmo quando nele foi executada a aplicação com otimização. Pode-se concluir a partir dos testes que quando for necessário avaliar o desempenho das aplicações desenvolvidas para a plataforma Android, as medições não devem ser baseadas apenas nos resultados do simulador.

4 CONCLUSÕES

Este trabalho apresentou uma aplicação com realidade aumentada utilizando coordenadas geográficas para registro dos objetos virtuais, chamados de pontos de interesse. A impressão de aumento de realidade é garantida através do uso da câmera de vídeo, na qual os objetos virtuais são desenhados acima das imagens da câmera. A interação da realidade com a virtualidade ocorre através da movimentação do dispositivo, acionando a bússola e o acelerômetro. Essas funcionalidades resultam no cumprimento de todos os objetivos propostos para o trabalho.

Também foi entregue por este trabalho um conjunto de funcionalidades que permitem ao desenvolvedor de aplicativos de realidade aumentada, testar seu desenvolvimento no simulador do Android. A primeira funcionalidade permite que as imagens de uma câmera externa sejam disponibilizadas pelo computador e capturadas pelo simulador. A segunda funcionalidade permite enviar ao simulador os valores de diversos tipos de sensores suportados pela plataforma como acelerômetro e bússola.

A plataforma Android mostrou possuir grande potencial para desenvolvimento de aplicativos, através da API bastante robusta e das ferramentas de desenvolvimento e depuração que permitem inclusive depurar dentro de um dispositivo. A transparência é uma característica nessa plataforma, que pôde ser observada ao utilizar as APIs dos serviços de localização e de sensores, toda interação com o *hardware* do dispositivo foi abstraída pela plataforma.

O aplicativo desenvolvido nesse trabalho apresentou funcionalidades semelhantes às dos trabalhos correlatos. Dentre elas pode-se destacar a determinação dos pontos de interesse através de coordenadas geográficas, acelerômetro e bússola; a visualização da realidade através da câmera e da virtualidade através de objetos desenhados em um sistema em 3D; e a obtenção dos pontos de interesse através de um servidor web.

Por fim, este trabalho apresentou um conceito inovador de interação da realidade com a virtualidade através de dispositivos móveis em uma das plataformas emergentes. Gerando como resultado uma aplicação funcional que se destaca não só pela imersão, mas também pela sua característica única de permitir executar no simulador, tornando a aplicação acessível aos desenvolvedores que não possuem o dispositivo.

4.1 EXTENSÕES

Durante o desenvolvimento foram verificadas algumas possíveis melhorias que não foram contempladas neste trabalho.

A técnica de desenho utilizada dentro da OpenGL para esse trabalho foi a de vértices básicos por ser a única compatível com todos os dispositivos da plataforma. Como melhoria sugere-se adicionar as duas outras técnicas identificadas na sessão 2.2.5, VBO e `GL_OES_draw_texture`, adicionando o teste na aplicação para verificar qual a melhor técnica para o dispositivo que está rodando o aplicativo.

A correção de problemas como o desenho do texto na OpenGL também apresenta uma melhoria, de forma a torná-lo compatível com o *depth test* e posicioná-lo corretamente no espaço de pintura. Outra maneira seria desenhar os textos em uma camada de `Canvas` que já possui tais recursos. Ainda dentro da OpenGL sugere-se desenhar texturas com imagens que representam os pontos de interesse com a possibilidade de animações e até a utilização de modelos em 3D, da mesma forma que o Layar (ver sessão 2.3). Outra possível extensão seria a de tratar o toque do objeto na imersão, simulando um toque virtual no objeto virtual e alterando, por exemplo, a sua forma ou posição.

Dentro do conceito de realidade aumentada existe ainda o registro dos pontos de interesse através de marcações impressas ou códigos de barras. Sugere-se explorar tais recursos de forma semelhante ao projeto AndAR (2010).

Por último sugere-se a criação de uma página de cadastro de pontos de interesse no servidor *web* com a possibilidade de adicionar a sua localização geográfica, imagens e modelos em 3D.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDAR. **AndAR, Android augmented reality**. [S.l.], 2010. Disponível em: <http://code.google.com/p/andar/>. Acesso em: 02 nov. 2010.

ANDROID DEVELOPERS. **OpenGL:** how to use gluUnProject on Android? [S.l.], 2010a. Disponível em: <http://groups.google.com/group/android-developers/browse_thread/thread/9d2bf53e3a798cb6>. Acesso em: 05 nov. 2010.

_____. **Merge gives odd z order hiding between custom views**. [S.l.], 2010b. Disponível em: <http://groups.google.com/group/android-developers/browse_thread/thread/9c335071c7919d80/827167f6a3dc08ee?pli=1#>. Acesso em: 27 out. 2010.

ARTAG. [S.l.], 2010. Disponível em: <http://www.artag.net/index.html>. Acesso em: 27 mar. 2010.

ARTOOLKIT. Washington, 2010. Disponível em: <http://www.hitl.washington.edu/artoolkit/>. Acesso em: 27 mar. 2010.

AZUMA, Ronald T. A Survey of augmented reality. **Presence**: Teleoperators And Virtual Environments, [S.l.], v. 6, n. 4, p. 355-385. Aug. 1997. Disponível em: <http://www.cs.unc.edu/~azuma/ARpresence.pdf>. Acesso em: 13 mar. 2010.

BIMBER, Oliver; RASKAR, Ramesh. **Spatial augmented reality merging real and virtual worlds.** [S.l.]: A K Peters LTD, 2005. Disponível em: <http://www.uni-weimar.de/medien/ar/SpatialAR/download.php>. Acesso em: 13 mar. 2010.

CALIFE, Daniel. **Robot ARena**: uma infra-estrutura para o desenvolvimento de jogos com realidade aumentada espacial. 2008. 110 f. Dissertação (Mestrado em Engenharia Elétrica) - Escola Politécnica da Universidade de São Paulo, São Paulo. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-28032008-162730/>. Acesso em: 26 mar. 2010.

GOOGLE. **Audio and video.** [S.l.], 2010a. Disponível em: <http://developer.android.com/guide/topics/media/index.html>. Acesso em: 20 mar. 2010.

_____. **Dalvik virtual machine internals.** [S.l.], 2010b. Disponível em: <http://developer.android.com/videos/index.html#v=ptjedOZEXPM>. Acesso em: 20 mar. 2010.

_____. **Design for performance**. [S.l.], 2010c. Disponível em: <http://developer.android.com/guide/practices/design/performance.html>. Acesso em: 13 set. 2010.

GOOGLE. **Graphics.** [S.l.], 2010d. Disponível em: <http://developer.android.com/guide/topics/graphics/index.html>. Acesso em: 15 ago. 2010.

_____. **Installing the SDK**. [S.l.], 2010e. Disponível em: <http://developer.android.com/sdk/installing.html>. Acesso em: 01 out. 2010.

_____. **Intents and intent filters.** [S.l.], 2010f. Disponível em: <http://developer.android.com/guide/topics/intents/intents-filters.html>. Acesso em: 20 mar. 2010.

_____. **Location and maps**. [S.l.], 2010g. Disponível em: <http://developer.android.com/guide/topics/location/index.html>. Acesso em: 09 set. 2010.

_____. **LocationManager**. [S.l.], 2010h. Disponível em: <http://developer.android.com/reference/android/location/LocationManager.html>. Acesso em: 09 set. 2010.

_____. **Obtaining user location**. [S.l.], 2010i. Disponível em: <http://developer.android.com/guide/topics/location/obtaining-user-location.html>. Acesso em: 09 set. 2010.

_____. **One screen turn deserves another**. [S.l.], 2010j. Disponível em: <http://android-developers.blogspot.com/2010/09/one-screen-turn-deserves-another.html>. Acesso em: 29 set. 2010.

_____. **SensorManager**. [S.l.], 2010k. Disponível em: <http://developer.android.com/reference/android/hardware/SensorManager.html>. Acesso em: 30 set. 2010.

_____. **Sprite method test readme**. [S.l.], 2010l. Disponível em: <http://code.google.com/p/apps-for-android/source/browse/trunk/SpriteMethodTest/README.TXT>. Acesso em: 16 ago. 2010.

_____. **Spritetext -** API demos. [S.l.], 2010m. Disponível em: <http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/graphics/spritetext/index.html>. Acesso em: 10 out. 2010.

_____. **USB driver for Windows**. [S.l.], 2010n. Disponível em: <http://developer.android.com/sdk/win-usb.html>. Acesso em: 01 out. 2010.

_____. **User interface.** [S.l.], 2010o. Disponível em: <http://developer.android.com/guide/topics/ui/index.html>. Acesso em: 15 ago. 2010.

_____. **What is Android?** [S.l.], 2010p. Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>. Acesso em: 13 mar. 2010.

GOOGLE. **Writing real-time games for Android.** [S.l.], 2010q. Disponível em: <http://developer.android.com/videos/index.html#v=U4Bk5rmIpic>. Acesso em: 15 ago. 2010.

GIS FAQ. **Great circle distance between 2 points**. [S.l.], 2010. Disponível em: <http://www.movable-type.co.uk/scripts/gis-faq-5.1.html>. Acesso em: 15 out. 2010.

GRAPH TECH. **OpenGL ES architecture on Android**. [S.l.], 2010. Disponível em: <http://blog.graphtech.co.il/opengl-es-architecture-on-android/>. Acesso em: 08 set. 2010.

GUJ. **A classe java.lang.String**. São Paulo, 2010. Disponível em: <http://www.guj.com.br/article.show.logic?id=103>. Acesso em: 13 set. 2010.

IMASTERS. **Entendendo as classes String, StringBuilder e StringBuffer**. [S.l.], 2010. Disponível em: <http://imasters.uol.com.br/artigo/7131/java/entendendo_as_classes_string_stringbuilder_e_stringbuffer/>. Acesso em: 13 set. 2010.

KIRNER, Cláudio; SISCOUTTO, Robson. Fundamentos de realidade virtual e aumentada. In: _____ (Ed.). **Realidade virtual e aumentada**: conceitos, projetos e aplicações. Porto Alegre: SBC, 2007. cap. 1, p. 2-21.

LAYAR BV. **An overview of the Layar platform**. Amsterdam, 2010a. Disponível em: <http://layar.com/create/platform-overview/>. Acesso em: 27 mar. 2010.

_____. **Layar´s visit to the US –** first stop: New York. Amsterdam, 2010b. Disponível em: <http://site.layar.com/company/blog/layars-visit-to-the-us-first-stop-new-york/>. Acesso em: 03 set. 2010.

_____. **The next wave of interactive augmented reality experiences**. Amsterdam, 2010c. Disponível em: <http://site.layar.com/company/blog/the-next-wave-of-interactive-augmented-reality-experiences/>. Acesso em: 01 out. 2010.

_____. **What is Layar?** Amsterdam, 2010d. Disponível em: <http://layar.com/download/layar/>. Acesso em: 13 mar. 2010.

LINUX GRAPHICS. **Writing real time games for Android**. Pequim, 2010. Disponível em: <http://www.linuxgraphics.cn/android/write_real_time_for_android.html>. Acesso em: 08 set. 2010.

MAGNITUDEHQ. **Magnitudehq**. Toulouse, 2010a. Disponível em: <http://code.google.com/p/magnitudehq/>. Acesso em: 01 out. 2010.

_____. **Project Magnitude**. Toulouse, 2010b. Disponível em: <http://www.magnitudehq.com/>. Acesso em: 01 out. 2010.

_____. **Le rapport de fin de projet**. Toulouse, 2010c. Disponível em: <http://code.google.com/p/magnitudehq/wiki/RapportFinProjet>. Acesso em: 01 out. 2010.

MOBILE TECH WORLD. **HTC Desire review**. [S.l.], 2010. Disponível em: <http://www.mobiletechworld.com/2010/05/17/htc-desire-review/>. Acesso em: 26 out. 2010.

NGA. **World geodetic system**. Washington, 2010. Disponível em: <https://www1.nga.mil/ProductsServices/GeodesyGeophysics/WorldGeodeticSystem/Pages/default.aspx>. Acesso em: 15 out. 2010.

NOVABOX. **iPhone ray picking gluUnProject sample**. [S.l.], 2010. Disponível em: <http://blog.nova-box.com/2010/05/iphone-ray-picking-glunproject-sample.html>. Acesso em: 05 nov. 2010.

NOVODA. **Motion sensors and orientation in Android**. [S.l.], 2010. Disponível em: <http://novoda.com/2009/05/02/motion-sensors-and-orientation-in-android/>. Acesso em: 30 ago. 2010.

OPEN INTENTS. **Sensor Simulator for simulating sensor data in real time**. [S.l.], 2010a. Disponível em: <http://code.google.com/p/openintents/wiki/SensorSimulator>. Acesso em: 21 out. 2010.

_____. **SensorSimulatorClient.java**. [S.l.], 2010b. Disponível em: <http://code.google.com/p/openintents/source/browse/trunk/sensorsimulator/SensorSimulatorSettings/src/org/openintents/sensorsimulator/hardware/SensorSimulatorClient.java>. Acesso em: 12 out. 2010.

ORACLE. **Java native interface**. Califórnia, 2010. Disponível em: <http://download.oracle.com/javase/1.5.0/docs/guide/jni/>. Acesso em: 08 set. 2010.

PDANET. **USB tether, Bluetooth DUN for Android**. [S.l.], 2010. Disponível em: <http://www.junefabrics.com/android/download.php>. Acesso em: 01 out. 2010.

SUN MICROSYSTEMS. **The class file format**. Palo Alto, 2010. Disponível em: <http://java.sun.com/docs/books/jvms/second_edition/html/ClassFile.doc.html#80959>. Acesso em: 27 mar. 2010.

SCHEMBERGER, Elder E.; FREITAS, Ivonei; VANI, Ramiro. Plataforma Android. **Jornal Tech**, [S.l.], n. 1, não paginado, ago. 2009. Disponível em: <http://www.jornaltech.com.br/wp-content/uploads/2009/09/Artigo_Android.pdf>. Acesso em: 13 mar. 2010.

TIEXPERT. **Get e set, métodos acessores**. [S.l.], 2010. Disponível em: <http://www.tiexpert.net/programacao/java/get-set.php>. Acesso em: 14 set. 2010.

TOM GIBARA. **Live camera previews in Android**. [S.l.], 2010. Disponível em: <http://www.tomgibara.com/android/camera-source>. Acesso em: 20 out. 2010.

TOPOGRAFIX. **GPX:** the GPS Exchange format. Massachusetts, 2010. Disponível em: <http://www.topografix.com/gpx.asp>. Acesso em: 09 set. 2010.

TU GRAZ. **Handheld augmented reality**. Graz, 2010a. Disponível em: <http://studierstube.icg.tu-graz.ac.at/handheld_ar/>. Acesso em: 27 mar. 2010.

_____. **Studierstube augmented reality project**. Graz, 2010b. Disponível em: <http://studierstube.icg.tu-graz.ac.at/>. Acesso em: 27 mar. 2010.

_____. **Studerstube es**. Graz, 2010c. Disponível em: <http://studierstube.icg.tu-graz.ac.at/handheld_ar/stbes.php>. Acesso em: 27 mar. 2010.

WIKIPEDIA. **Android dev phone**. San Francisco, 2010a. Disponível em: <http://en.wikipedia.org/wiki/Android_Dev_Phone#Android_Dev_Phone_1>. Acesso em: 08 set. 2010.

_____. **Keyhole markup language**. San Francisco, 2010b. Disponível em: <http://pt.wikipedia.org/wiki/Keyhole_Markup_Language>. Acesso em: 09 set. 2010.

_____. **Observer**. San Francisco, 2010c. Disponível em: <http://pt.wikipedia.org/wiki/Observer>. Acesso em: 09 set. 2010.

_____. **OpenGL ES**. San Francisco, 2010d. Disponível em: <http://pt.wikipedia.org/wiki/OpenGL_ES>. Acesso em: 15 ago. 2010.

_____. **Plumb-bob**. San Francisco, 2010e. Disponível em: <http://en.wikipedia.org/wiki/Plumb-bob>. Acesso em: 28 set. 2010.