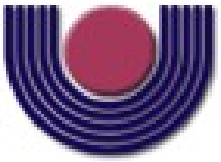


UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

Avaliação de aplicações gráficas em diferentes versões do sistema Android

André Specian Cardoso

CASCAVEL

2012

ANDRÉ SPECIAN CARDOSO

Avaliação de aplicações gráficas em diferentes versões do sistema Android

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel.

Orientador: Adair Santa Catarina

CASCAVEL

2012

ANDRÉ SPECIAN CARDOSO

Avaliação de aplicações gráficas em diferentes versões do sistema Android

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Adair Santa Catarina (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Edmar A. Bellorini
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Márcio S. Oyamada
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 17 de Outubro de 2012.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me capacitado cada dia, me dando forças para que eu pudesse vencer cada um dos desafios durante esta caminhada. Também a minha família que sempre esteve ao eu lado me apoiando e incentivando nos momentos mais difíceis, fazendo com que seguisse em frente sem desanimar.

Agradeço também a todos os professores que de alguma maneira me ajudaram no desenvolvimento do trabalho, em especial ao professor Adair que durante todo o ano me orientou e me ajudou a resolver todas os problemas encontrados.

Sou muito grato aos meus colegas/amigos de turma que por cinco anos caminhamos juntos, compartilhando muitos momentos de alegria, os quais fizeram com que os momentos de dificuldade se tornassem mais fáceis de serem superados.

Enfim, agradeço a todos que cruzaram o meu caminho e que de alguma forma me ajudaram a vencer mais um desafio.

Lista de Figuras

Sumário

Resumo

Com a popularização dos *smartphones* que utilizam o sistema Android e a velocidade com que atualizações deste sistema têm sido realizadas, rapidamente estes aparelhos tornam-se desatualizados. A indústria de dispositivos móveis atualizam suas linhas de produtos em intervalos cada vez menores e, portanto, deixam de oferecer atualizações para dispositivos lançados há pouco tempo. Em cada nova versão do sistema Android inclui-se suporte para novas funcionalidades, oriundas de *hardware* e *software*. Assim, maior capacidade de processamento é exigida pelo sistema, o que pode influenciar no desempenho de aplicações gráficas. Para verificar a compatibilidade e o desempenho em diferentes versões do sistema Android, desenvolveu-se um aplicativo gráfico que utiliza OpenGL ES 1.0. Este aplicativo consistiu num conjunto de seis testes que medem o consumo de CPU e o número de frames por segundo renderizados. Os testes utilizam diferentes efeitos gráficos e suas combinações: iluminação/sombreamento, *fog* e textura. Foram selecionadas duas versões do sistema Android para avaliação: *Gingerbread* v2.3.4 e *CyanogenMod Ice Cream Sandwich* v.4.0.4. Ambas foram instaladas num *smartphone* LG *Optimus Net*. Os testes realizados na versão 2.3.4 manipularam, na cena gerada, 100 objetos enquanto na versão 4.0.4 a cena foi composta por apenas 10 objetos. A versão 2.3.4 renderiza, em média, um número maior de *fps*: 45 *fps* contra 37 *fps* para a outra versão. Quanto à compatibilidade não houveram problemas; ambos os sistemas executaram corretamente a mesma aplicação, mostrando que o OpenGL ES v1.0 é igualmente suportado pelas duas versões avaliadas.

Palavras-chave: *smartphone*, desempenho, compatibilidade, OpenGL ES

Capítulo 1

Introdução

Inicialmente Android era apenas o nome de uma pequena empresa de *software* fundada em 2003 na Califórnia, EUA. Após 2 anos, a Google comprou essa pequena empresa contratando os principais desenvolvedores, fazendo com que surgissem boatos de que estaria interessada em entrar no mercado dos telefones móveis [1][2]. Esses boatos continuaram até 2008, ano em que a Google lançou a primeira versão do seu sistema operacional móvel, o Android 1.0, juntamente com o primeiro dispositivo a utilizá-lo, o HTC *Dream* (G1), tornando os boatos verdadeiros [1][2][3][4][5]. Desde então o uso do Android cresceu de tal maneira que, em apenas 4 anos, é utilizado por mais de 50% dos assinantes de *smartphones* dos EUA, apresentando crescimento de 3,7% desde o final de 2011 até Março de 2012 [6][7].

Acredita-se que o principal motivo para que o Android fizesse tanto sucesso foi a formação da OHA (*Open Handset Alliance*) no final de 2007. A OHA é uma aliança formada por grupos de desenvolvedores de *software*, fabricantes e operadores de celulares, entre outros, como a HTC, Qualcomm, Motorola e Nvidia, que colaboraram para desenvolver padrões abertos para dispositivos móveis [3][8][9]. Outro motivo é a sua capacidade de executar vários programas ao mesmo tempo (multitarefa), sua capacidade de utilizar vários processadores (multiprocessamento), uso de *widgets* (atalhos rápidos para os aplicativos), personalização da interface, armazenamento interno e/ou uso em cartões microSD, suporte a aplicações 2D e 3D, entre outras [10].

Após o seu lançamento, em 2008, foram desenvolvidas várias versões do Android. Cada versão adicionou novas funcionalidades para a plataforma, principalmente para desenvolvedores de jogos. A versão 1.5 (*Cupcake*) adicionou suporte para bibliotecas nativas em aplicativos Android. A versão 1.6 (*Donut*) adicionou o suporte a telas com tamanhos diferentes. A versão 2.0 (*Eclair*) trouxe melhor resolução para a tela e adicionou suporte para multi-touch. A versão 2.2 (*Froyo*) melhorou ainda mais a resolução e adicionou suporte a

tecnologia Flash, sendo uma das versões mais utilizadas até hoje. A versão 2.3 (*Gigerbread*) acrescentou um novo coletor de lixo e a troca de dados e conexões sem fio entre dois dispositivos (NFC - *Near Field Communication*). A versão 3.0 (*Honeycomb*) foi otimizada para *tablets* adicionando aceleração de vários formatos de vídeos via *hardware* e suporte aos novos processadores de vários núcleos. A versão 4.0 (*Ice Cream Sandwich*) foi desenvolvida para unificar os *tablets* e os *smartphones* em uma única versão, adicionando o controle de tráfego de Internet, o destravamento de tela por reconhecimento de face, entre outros [3][11]. A versão 4.1 (*Jelly Bean*), a última lançada até hoje, foi anunciada em junho de 2012. Esta versão tem como objetivo melhorar a funcionalidade e desempenho da interface do usuário, fazendo uso do *Project Butter*, que consiste em modificar as bases do *software* e adicionar alguns recursos que vão melhorar a resposta e a utilização dos recursos. Outra novidade é a utilização do temporizador de sincronia vertical (*vsync*) prolongado para que o processador possa detectar o que acontece na tela [12].

1.1 Open Handset Alliance (OHA)

Em 5 de novembro de 2007, a Google e mais 33 empresas ligadas ao desenvolvimento de *softwares*, fabricação de celulares, fabricação de semicondutores, operadoras de celulares e companhias de comércio, uniram-se para criar a OHA. O objetivo do OHA é facilitar as inovações no mercado dos dispositivos móveis através de aplicações padronizadas, oferecendo aos consumidores uma experiência mais rica, mais barata e com maior portabilidade. Hoje em dia a OHA já possui mais de 80 empresas ligadas ao desenvolvimento do Android, dentre as principais estão a eBay, Intel, Nvidia, LG, Motorola, Samsung, Sony Ericsson, Toshiba e, é claro, a Google [1][3][8][9][13][14][15].

1.2 Um pouco mais sobre o Android

Android é um sistema operacional móvel baseado no *Kernel* Linux 2.6 sendo desenvolvido inicialmente pela empresa Android, antes mesmo da Google comprá-la [16]. Como a Google pretendia fazer um sistema operacional livre, ela liberou a maior parte do código fonte sob a licença Apache 2, o que quer dizer que qualquer um é livre para fazer sua própria aplicação, assim como os fornecedores podem criar extensões e personalizações para que seus produtos possam ser diferenciados [3][16]. Entretanto, para que uma nova aplicação seja compatível com o sistema operacional Android, essa deverá passar por um programa de

compatibilidade para assegurar a compatibilidade básica com o sistema; isso inclui a capacidade de ser executada em dispositivos diferentes e que utilizem diferentes versões do sistema [3]. Esse programa de compatibilidade, empregado em cada nova aplicação, garante que o sistema Android seja unificado, pois as aplicações não dependerão do dispositivo para serem executadas, dependerão apenas do sistema operacional.

O desenvolvimento de aplicações unificadas é uma das principais vantagens do Android, fazendo com que ele se torne padronizado e portável. Sabe-se que os aplicativos para os *smartphones* contribuíram para o sucesso desses dispositivos móveis; apesar das várias versões lançadas, as funcionalidades das versões anteriores podem ser executadas normalmente pelas versões mais recentes do Android [3][16].

Apesar do sucesso, o Android apresenta algumas desvantagens. A principal delas é ocasionada pela velocidade de lançamento das novas versões; apesar da compatibilidade entre elas, a constante atualização do sistema o torna rapidamente ultrapassado fazendo com que as empresas, que criam suas próprias interfaces de sistema, abandonem seus projetos e passem a trabalhar em novas aplicações e aparelhos para as novas versões do sistema. Assim, muitos aparelhos tornam-se ultrapassados pela falta de atualizações que atenderiam as modificações realizadas em versões mais novas [3]. Outra desvantagem é que muitas das funcionalidades desenvolvidas para versões mais recentes não funcionam nas anteriores, pois se utilizam de recursos não suportados, forçando os desenvolvedores a criarem códigos separados para as diferentes versões do sistema [3].

Um dos motivos para o Android evoluir tão depressa é a criação de aplicativos que exigem cada vez mais recursos, tanto do *hardware* como do *software*; por exemplo, execução de vídeos em alta definição, visualização e edição de imagens e, principalmente, os jogos 3D. Com isso a compatibilidade, em relação ao desempenho gráfico entre aplicativos criados para diferentes versões, pode ser afetada. Por exemplo, se uma aplicação executada no Android 3.0 utilizar recursos disponíveis apenas para essa versão, com certeza, não será executada em um aparelho que utilize o Android 2.1 [17][18].

Uma solução que pode ser adotada para eliminar ou diminuir a incompatibilidade entre as versões é a utilização dos recursos do OpenGL ES no desenvolvimento das novas aplicações; contudo deve-se escolher uma das versões do OpenGL ES, a 1.0 ou a 2.0, pois as duas versões também não são compatíveis entre si. A causa dessa incompatibilidade é a utilização de *pipelines* fixos na versão 1.0 enquanto a versão 2.0 utiliza *pipelines*

programáveis, onde boa parte da renderização é feita pelo programador, podendo assim criar aplicações gráficas com mais detalhes [19][20]. Neste trabalho será possível encontrar respostas para algumas dessas questões sobre compatibilidade de aplicativos gráficos.

1.3 Programa de Compatibilidade

O Programa de Compatibilidade apresenta os detalhes técnicos da plataforma Android, e também disponibiliza ferramentas usadas pelos desenvolvedores para garantir que os aplicativos sejam executados em uma variedade de dispositivos. O Android SDK (*Software Development Kit*) fornece meios para que os desenvolvedores indiquem claramente os recursos do dispositivo exigidos por suas aplicações. Assim, somente aplicações compatíveis com o dispositivo estarão disponíveis, para *download*, na loja virtual *Google Play*.

O Programa de Compatibilidade consiste em três componentes principais que, agrupados, resultam na avaliação da compatibilidade entre um dispositivo e uma aplicação [21]. São eles:

- O código fonte do Android;
- O Documento de Definição de Compatibilidade (CDD - *Compatibility Definition Document*)
- O Conjunto de Testes de Compatibilidade (CTS - *Compatibility Test Suite*)

O CTS é um conjunto de testes de nível comercial, livre e disponível para *download*; ele representa o mecanismo de compatibilidade. Depois de baixado, ele é executado no desktop e executa os casos de teste diretamente em dispositivos ou emuladores. O CTS é um conjunto de testes de unidade concebido para ser integrado no fluxo de trabalho de construção de um dispositivo. Sua intenção é revelar incompatibilidades desde o início e garantir que o *software* seja compatível em todo o processo de desenvolvimento [21].

O CDD é um documento detalhado da definição de compatibilidade, representando o aspecto “político” do programa. Cada versão da plataforma Android possui um CDD. Seu principal objetivo é esclarecer os requisitos, eliminando ambiguidades. Agindo como um hub, o CDD referencia outros conteúdos, como a documentação do SDK, e fornece um *framework* no qual um código fonte possa ser utilizado de maneira que o resultado final seja um sistema compatível. [21]

1.4 Objetivos

Este trabalho tem como objetivo verificar a compatibilidade e o desempenho em diferentes versões do sistema Android, ao executar aplicativos gráficos. Foram escolhidas duas versões do sistema: a *Gingerbread* v2.3.4 e a *Cyanogenmod* 9 - *Ice Cream Sandwich* v4.0.4 (*Release Candidate* - *Unofficial*), executadas em um *smartphone* da marca LG, modelo *Optimus Net*. A primeira é a versão original do Android para o aparelho em questão; a segunda versão é uma *Custom ROM* desenvolvida pelo grupo *CyanogenMod* a partir do Android v4.0.4.

Para alcançar o objetivo proposto, desenvolveu-se um aplicativo gráfico que utiliza a biblioteca OpenGL ES v1.0. A correta execução do aplicativo foi utilizada para avaliar a compatibilidade entre as duas diferentes versões do sistema Android; o número de FPS (*Frames* por segundo) e o percentual de uso da CPU foram utilizados para avaliar as diferenças de desempenho entre as versões.

1.5 Organização do Texto

Esta seção apresenta a estrutura do trabalho, listando seus capítulos e uma breve descrição do conteúdo abordado em cada um deles.

O Capítulo 2 apresenta a revisão de literatura onde são abordados o OpenGL e o OpenGL ES, que são APIs utilizadas no desenvolvimento de aplicações gráficas tridimensionais; a segunda API é especifica para sistemas embarcados, como os *smartphones*. Neste capítulo também explicou-se o que são as *Custom ROMs* e mostrou-se como fazer uma pequena aplicação para o sistema Android, para iniciantes (Apêndice A).

O Capítulo 3 descreve o material utilizado no desenvolvimento do trabalho: o *smartphone* usado nos testes, o ambiente de programação e a aplicação desenvolvida para realizar os testes de compatibilidade e desempenho.

O Capítulo 4 mostra, em detalhes, quais foram os parâmetros usados nos testes, os resultados obtidos e uma breve avaliação dos mesmos.

O Capítulo 5 apresenta a comparação dos resultados obtidos em cada versão de Android avaliada, finalizando com a conclusão sobre o desempenho em cada uma delas. Também são citadas as principais dificuldades encontradas durante a realização deste trabalho.

Capítulo 2

Revisão de Literatura

2.1 OpenGL (*Open Graphics Library*)

OpenGL é uma API (*Application Programming Interface*) utilizada para criação de sistemas que utilizam computação gráfica em aplicações 2D e, principalmente, aplicações 3D, como jogos e ferramentas de modelagem [21][22]. Em outras palavras, OpenGL é uma interface de *software* para *hardware* gráfico. [23]

O OpenGL também pode ser visto como uma biblioteca que possui mais de 250 chamadas para funções distintas, que fornecem acesso a praticamente todos os recursos gráficos do *hardware* de vídeo. Essas funções são utilizadas no desenho de cenas complexas a partir de simples primitivas (pontos, linhas e polígonos). Assim, o programador só precisa especificar quais são as operações e os objetos envolvidos na produção das cenas [22][23] [24].

O OpenGL é uma máquina de estados; cada estado representa uma configuração para o OpenGL. Por exemplo a cor, a transparência dos objetos em cena, o modelo de iluminação empregado, o efeito de *fog* (neblina), etc., determinam um estado válido para todos os objetos em cena. Por exemplo, enquanto a cor não mudar, todos os objetos serão desenhados com a mesma cor. [21][22][25]

O núcleo do OpenGL é conhecido como *rendering pipeline*. Ele é responsável pela preparação e manipulação dos dados relacionados com as coordenadas dos vértices dos objetos e também pelos efeitos que serão aplicados sobre a cena. Os principais elementos do *rendering pipeline* são [22][25] :

- *Lista de Exibição*: Lista contendo todos os dados que serão exibidos.

- *Avaliadores*: Todas as primitivas geométricas descritas por vértices. Curvas e superfícies podem ser descritas por fórmulas.
- *Operações pré-vértices:* Converte os vértices em primitivas de desenho. Os vértices são armazenados numa matriz 4x4 para que possam ser mapeados da coordenada 3D para as coordenadas de tela. Nesta etapa também são feito os cálculos de iluminação, textura e materiais, de acordo com a configuração atual.
- *Montagem de primitivas:* Elimina as partes da cena que estão fora do plano. Essa etapa também faz os cálculos de perspectiva, fazendo com que os objetos que estão mais longe pareçam menores do que os que estão mais próximos.
- *Operações com pixels:* Primeiramente os pixels são descompactados de acordo com seu número de componentes. Posteriormente os pixels têm sua escala ajustada e seu formato final é calculado, concluído isto, são enviados para a fase de rasterização.
- *Montagem de textura:* Aplica as imagens de textura nos objetos, para que pareçam mais realistas.
- *Rasterização:* Transforma todos os dados em fragmentos. Cada fragmento é um quadrado que corresponde a pixels no quadro de armazenamento auxiliar *(framebuffer).* Padrões de linhas e polígonos, largura de linhas, tamanho do ponto, sombreamento (*shading*)*,* cálculos de cobertura e *antialiasing* também são levados em conta nessa fase. Cores e valores de profundidade são considerados para cada fragmento.
- *Operação sobre fragmentos:* São aplicadas as operações finais nos fragmentos, como recortes, operações com máscaras de bits e mistura (*blending*). Finalizado o processo os pixels são desenhados na tela de exibição.

Em geral existem quatro versões do OpenGL, as versões 1, 2, 3 e 4, sendo que todas possuem subversões. A versão mais recente, até o momento, é a versão 4.3, lançada em 6 de agosto de 2012. As versões 1.x são compatíveis apenas entre si, não sendo compatíveis com as outras versões. Essa incompatibilidade ocorre pelo fato do OpenGL 1.x utilizar um *pipeline* fixo, o que limita muito o desenvolvimento, pois restringe os desenvolvedores a utilizarem uma determinada quantidade de estados do *pipeline* para gerar a saída, o que geralmente resultava em gráficos com visuais simétricos [24].

Com a criação das GPU's (*Graphics Processing Units*) programáveis (*pipeline* programável) e o uso de *shaders*, várias técnicas de processamento gráfico podem ser implementadas diretamente na placa de vídeo, como os cálculos de transformações geométricas e o cálculo de iluminação. Com isso o processo gráfico se tornou muito mais rápido, pois enquanto a GPU faz os cálculos intensivos de forma mais rápida que a CPU, esta se encarregava de processar outros dados, como carregar os próximos objetos em cena que irão sofrer transformações e desenhar os objetos que já passaram pelo processo. [26]

2.2 OpenGL ES (*OpenGL for Embedded Systems*)

O OpenGL ES é uma API de baixo nível para sistemas embarcados criada com base no OpenGL padrão, fornecendo baixo nível de programação para aplicações tanto de *software* como de *hardware*. Desta maneira o desenvolvimento de aplicações gráficas, como jogos e animações tridimensionais, torna-se mais fácil e possibilita o incremento no nível de detalhes dos objetos gráficos apresentados.

Existem três versões do OpenGL ES que são utilizadas no Android, as versões 1.0, 1.1 e 2.0, sendo esperada a versão 3.0 para este ano[27][28][29]. Cada versão foi desenvolvida com base em uma versão diferente do OpenGL; a versão 1.0 do OpenGL ES foi criada com base na especificação do OpenGL 1.3 acrescentando poucas funcionalidades, uma delas em relação a sintaxe. Por exemplo, a criação de objetos que, no OpenGL, necessita elencar seus vértices dentro das cláusulas glBegin e glEnd; já no OpenGL ES 1.0 foram retiradas estas cláusulas [19][27].

O OpenGL ES 1.1 foi desenvolvido com base na especificação do OpenGL 1.5 adicionando alguns recursos, como suporte para multitexturas, buffers de vértices para os objetos e maior controle de processamento dos pontos dos objetos, melhorando a qualidade de imagem e aumentando o desempenho [19][27].

Com base no OpenGL 2.0 foi desenvolvido o OpenGL ES 2.0. Nesta versão grande parte do *pipeline* fixo foi substituído por um programável; este possibilitou aos desenvolvedores implementar boa parte da renderização, permitindo a criação de aplicações gráficas mais detalhadas [19][27].

Se compararmos a figura 2.1 com a figura 2.2, que mostram os diagramas do *pipeline* fixo e programável, respectivamente, podemos encontrar porque o *pipeline* programável torna possível aos desenvolvedores criarem cenas tridimensionais mais detalhadas. Praticamente

todas as operações que tratam dos cálculos de transformações, texturas e iluminação foram substituídos por *shaders*, que são um conjunto de programas usados para cálculos de renderização por uma GPU, como transformações geométricas, iluminação, sombreamento, etc. [30][31][32][33][34].

Como visto na figura 2.2, existem dois tipos de *shaders*: o *Vertex Shader* e o *Fragment Shader*. O *Vertex Shader* substitui a seção de texturização e iluminação do pipeline gráfico, realiza operações tradicionais, como transformar uma cena 3D (coordenadas de mundo) em 2D (coordenadas de tela/dispositivo) e aplica rotações e translações nos objetos, possibilitando modificar a estrutura dos objetos em tempo de execução [26][31]. O *Fragment Shader*, ou *Pixel Shader*, trabalha especificamente com os pixels de cada objeto e é responsável pelo processamento de efeitos de renderização como iluminação e sombreamento dos objetos, interpolação de valores, aplicação de textura e *fog* (nevoeiro), modificando a cor dos pixels [26][32][35][36].

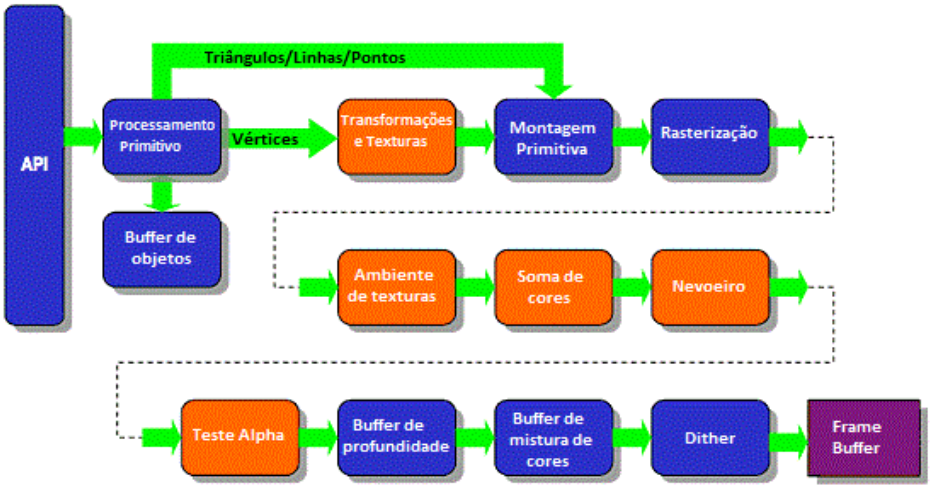


Figura 2.1 – *Pipeline* fixo (Adaptado de Khronos Group [19])

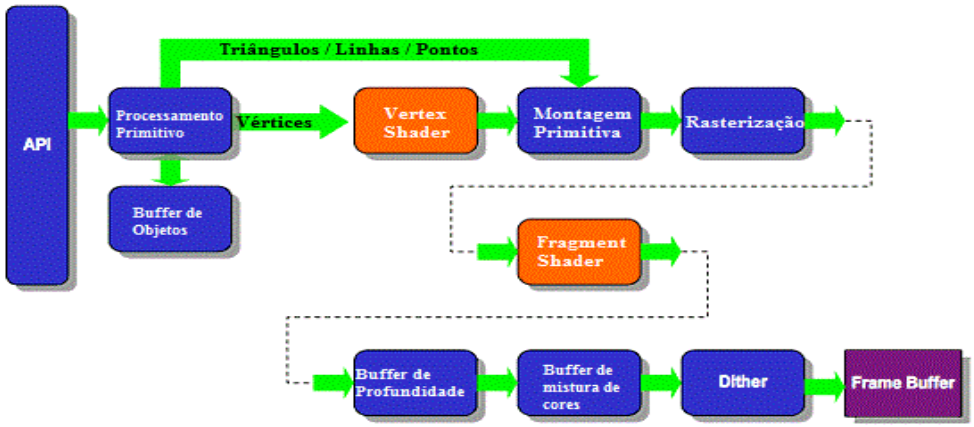


Figura 2.2 – *Pipeline* programável (Adaptado de Khronos Group [19])

2.3 ROMs customizadas para Android (Custom ROMs)

ROMs customizadas nada mais são do que uma versão personalizada de alguma versão do Android. Estas customizações podem ser construídas a partir das versões que vieram dos fabricantes como podem ser construídas a partir do zero, com base no código do Android disponibilizado pela Google no AOSP (Android Open Source Project). [37]

Ao serem instaladas, essas versões customizadas substituem a versão do Android instalada pela fábrica no *smartphone*. Por isso é recomendado fazer uma boa pesquisa sobre cada uma delas para conhecer seus defeitos e os procedimentos de instalação. Duas das Custom ROMs mais utilizadas são a MIUI (Figura 2.3) e a *CyanogenMod* (2.4), que possuem versões para vários modelos de *smartphones*. Ambas são desenvolvidas sobre regras bastante claras e apresentam suporte por parte de seus desenvolvedores. [37]

Figura 2.3 – Interface do *Custom ROM MIUI v*4.0

Figura 2.4 – Interface do *Custom ROM CyanogenMod* 9

Como cada fabricante de *smartphone* geralmente constrói um *hardware* diferente para cada um dos aparelhos fabricados, não se pode ter certeza de que uma *Custom ROM* funcionará corretamente; alguns componentes do *hardware* como a câmera, o processador ou a memória RAM podem falhar. Por este motivo há a necessidade de se adaptar o sistema Android, alterando o seu código [38].

Assim, quando o usuário decide instalar uma *Custom ROM* em seu aparelho, ele deve prestar atenção e fazer as adaptações corretas no sistema para que não hajam problemas [38]. Outra preocupação está relacionada com a ROM escolhida, pois cada *Custom ROM* é fabricada com base na ROM original de cada aparelho, portanto, é provável que não seja possível utilizar uma mesma ROM em aparelhos diferentes, principalmente quando são de fabricantes diferentes [37].

Um dos motivos pelo qual muitos usuários preferem as *Custom ROM*s ao invés das ROMs originais de fábricas, é pelo simples fato de conseguir atualizações para seus aparelhos, como no caso do HTC G1, que recebeu atualizações oficiais somente até a versão 1.6 [38].

Uma das vantagens de ser utilizar as *Custom ROM*s é melhora no desempenho do sistema; o sistema Android se torna mais limpo ao ser desenvolvido em um código puro, fornecido pela Google. Os desenvolvedores aplicam apenas as modificações necessárias, como os drivers do dispositivo, fazendo com que o sistema execute adequadamente no aparelho. [37]

O código otimizado e a ausência de aplicativos desnecessários pré-instalados, fazem com que as *Custom ROMs* sejam consideradas melhores que o sistema original instalado. Porém, caso a instalação seja feita de modo incorreto, o sistema pode ser completamente arruinado. Por isso aconselha-se pesquisar antes de se trocar o sistema Android original por uma *Custom ROM*.

2.4 Criando uma aplicação para Android

Para criar um aplicativo para o sistema Android é necessário instalar e configurar um conjunto de ferramentas que inclui o *Java SE Development Kit* (JDK), um ambiente de desenvolvimento, o Android *Software Development Kit* (SDK) e o *Android Development Tools* (ADT)[16].

O desenvolvimento de aplicativos para Android podem ser feitos usando um Mac, um computador com Windows ou um computador com Linux. O Android é *open source* e tem como base o *Kernel* do Linux, o que significa que pode ser alterado livremente. Por ser um código livre, todas as ferramentas necessárias para o desenvolvimento de um aplicativo Android são gratuitas e estão disponíveis para *download* na Internet, facilitando o desenvolvimento de novos aplicativos[16].

O Android SDK utiliza o *Java SE Development Kit* (JDK). Caso o computador do usuário não possua o JDK instalado será necessário fazer seu *download* a partir do link: www.oracle.com/technetwork/java/JavaSE/downloads/index.html[16].

Um tutorial mostrando como instalar e configurar o ambiente de desenvolvimento, no sistema operacional *Microsoft Windows* 7, bem como um primeiro exemplo da aplicação Android é apresentado no apêndice A deste trabalho [16].

Capítulo 3

Materiais e Métodos

Neste capítulo serão apresentados os recursos utilizados na execução deste trabalho, assim como os procedimentos empregados.

3.1 Material utilizado

Para efetuar a comparação entre as duas versões de Android selecionadas, utilizou-se um mesmo modelo de *smartphone*, evitando que o desempenho do *hardware* interfira no resultado final dos testes. O modelo escolhido para os testes foi o *smartphone* LG-P690 *Optimus Net*, com a versão do Android 2.3.4 instalado pela fábrica, com a versão de *Kernel* 2.6.35.10, *chipset Qualcomm* MSM7227T, CPU 800 MHz ARM 11 *single-core*, GPU Adreno 200, 150 Mb de armazenamento, 512 Mb de RAM, 512 Mb de ROM e resolução 320 x 480 *pixels*. Este aparelho foi escolhido por ser facilmente atualizável para outras versões, não tendo a necessidade de utilizar dois aparelhos com versões distintas de sistemas Android.

A segunda versão avaliada foi a *custom* ROM *unofficial CyanogenMod* 9, com Android 4.0.4 e *Kernel* 2.6.35.14, disponível em http://www.mediafire.com/?2vp9425z39hto4b.

Durante o desenvolvimento foi utilizado o ambiente de desenvolvimento Eclipse Helios Service Release 2 juntamente com o Android SDK e AVD Manager para emular o sistema Android durante os testes.

3.2 Desenvolvimento da aplicação

Para alcançar o objetivo deste trabalho, que é o de verificar a compatibilidade e o desempenho gráfico entre duas diferentes versões do sistema Android, desenvolveu-se uma aplicação gráfica 3D, descrita a seguir.

A aplicação desenvolvida manipula vários objetos tridimensionais em uma mesma cena, aplicando a eles diversos efeitos como sombreamento, iluminação e texturas. Para averiguar o nível de compatibilidade entre as versões do Android foram criados vários testes, com níveis crescentes de complexidade; quanto maior o nível de complexidade, maior será o processamento necessário para a execução da aplicação. Desse modo foi possível estruturar uma classificação para avaliar o desempenho da aplicação gráfica desenvolvida, no sistema que está sendo testado.

A aplicação desenvolvida consiste em seis diferentes testes com níveis crescentes de complexidade, explicados em detalhes na próxima seção. Estes testes foram classificados em três níveis, onde são observados diferentes aspectos do sistema. Os três níveis do sistema são:

- Nível um: o único teste classificado neste nível é o teste 1, pois ele é o mais simples; este nível tem como objetivo principal analisar a capacidade das versões do Android selecionadas para suportar aplicações gráficas tridimensionais, ou seja, verificar como as versões do Android se comportam ao executarem aplicativos tridimensionais simples.

- Nível dois: Este nível é onde realmente começam os testes, pois é nele que são aplicados alguns dos efeitos. Este nível avalia a capacidade das versões do Android para executar uma aplicação tridimensional, sendo exigido alguns detalhes. Os testes que se encaixam neste nível são os testes 2, 3 e 4.

- Nível três: Neste último nível a aplicação é muito mais detalhada, aplicando diversos efeitos sobre os objetos ao mesmo tempo, criando uma cena um pouco mais realista e exigindo desempenho máximo do sistema. É aqui que as imagens são analisadas para a comparação de qualidade. Neste nível estão classificados os dois últimos testes, o teste 5 e o teste 6.

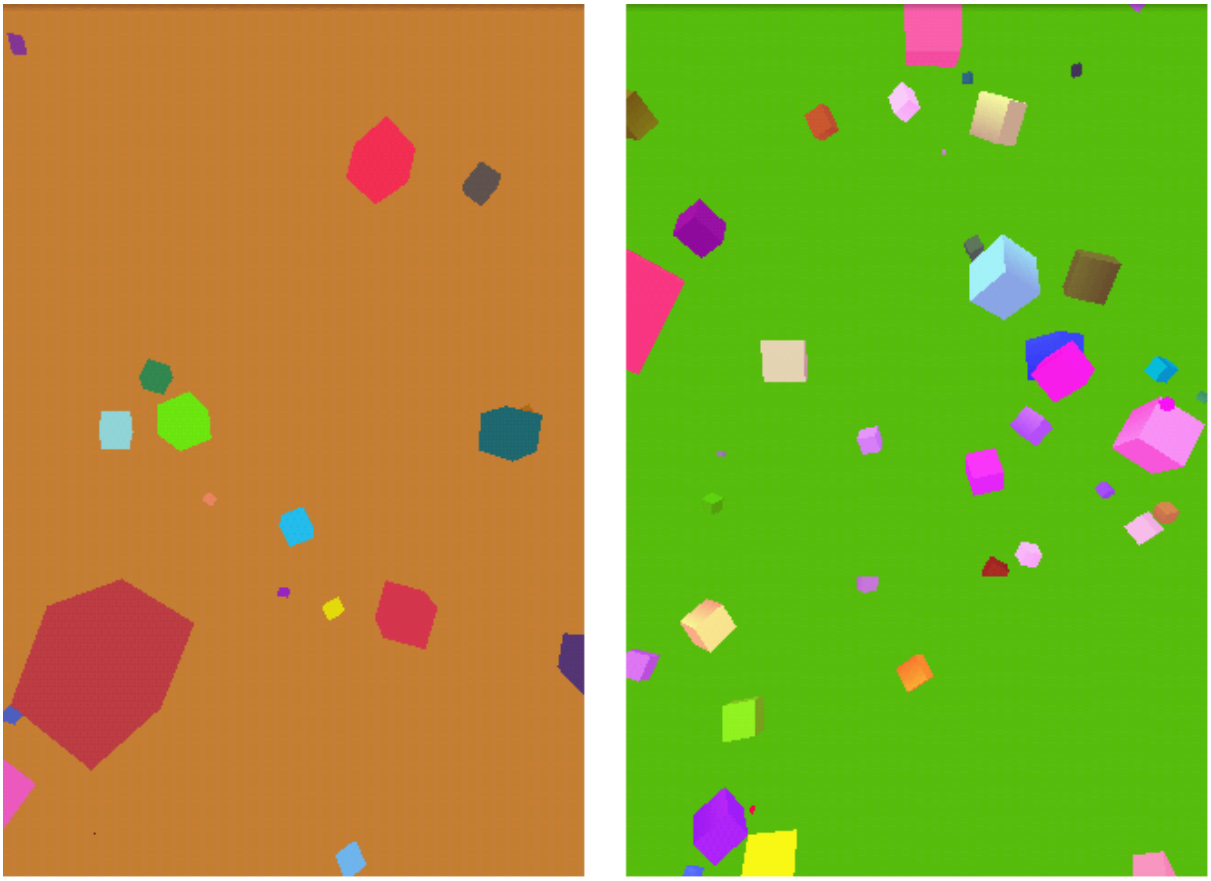
Para que a aplicação seja suportadas pelas diversas versões do Android já lançadas é necessário utilizar os recursos do OpenGL ES 1.0. Apesar de de existirem versões mais aprimoradas, elas são suportadas apenas pelas versões do sistema Android superiores a 2.2.

3.3 Desenvolvimento dos testes

Nesta seção serão explicados detalhadamente os seis testes desenvolvidos para ao sistema. Os testes foram desenvolvidos em separado, visando aumentar o controle e uma avaliação mais detalhada.

O teste 1 foi desenvolvido para verificar como as diferentes versões do Android testadas se comportam ao executarem aplicações gráficas tridimensionais. Nos objetos em cena não foram aplicados efeitos, eles apenas foram colorizados aleatoriamente, movendo-se também aleatoriamente pelo cenário (Figura 3.1 a).

No teste 2 foi exigido mais processamento. Neste teste foi aplicado apenas o efeito de iluminação/sombreamento; foram criadas cinco fontes de luz, tornando os objetos mais claros ao se aproximarem das luzes e mais escuros ao se afastarem. Assim como os objetos as luzes também têm posições e cores aleatórias (Figura 3.1 b).

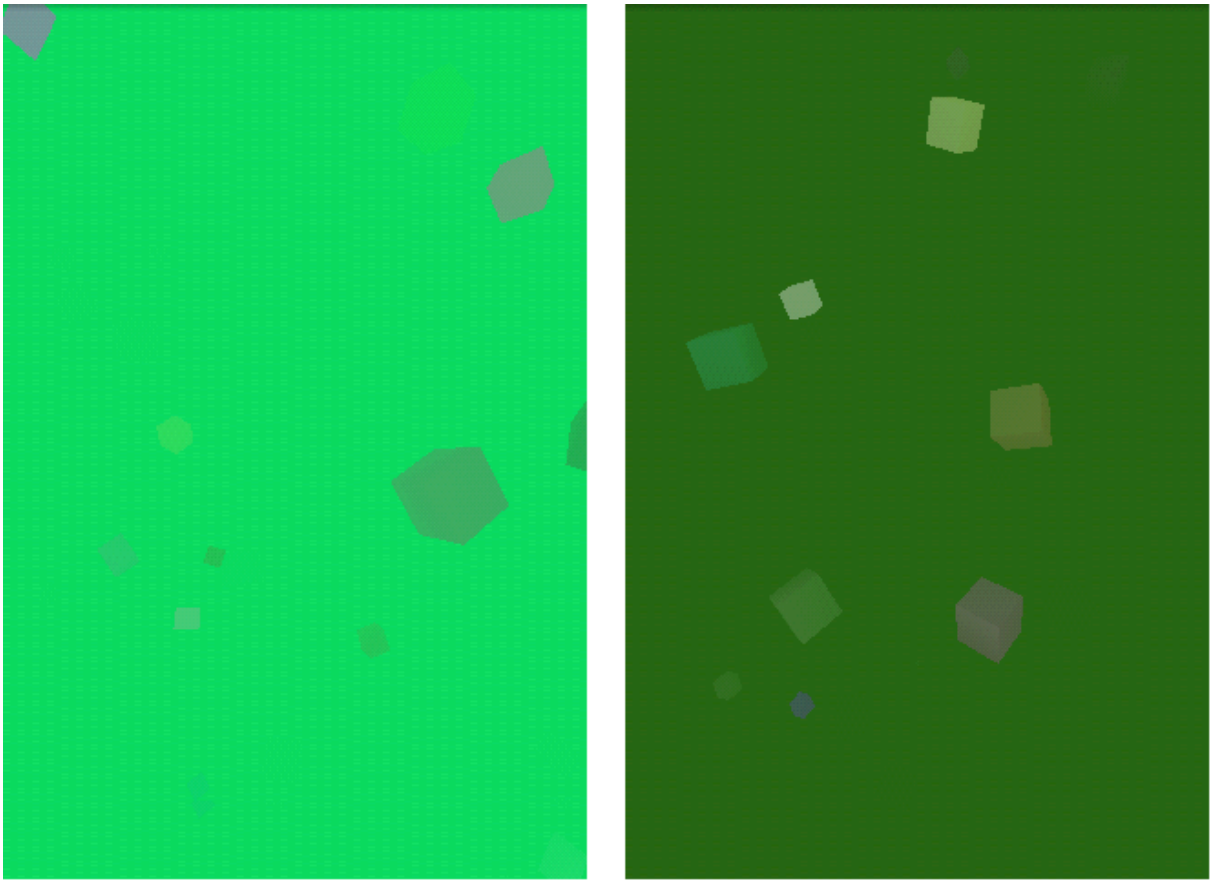


(a) (b)

Figura 3.1 – Telas capturadas durante a execução do teste 01 (a) e do teste 02 (b)

Para o teste 3 foi aplicado apenas o efeito *fog* (névoa) linear sobre a cena, retirando assim as fontes de luz do teste anterior. Este efeito tem como característica fazer com que os objetos que estão muito longe ou muito próximos da câmara sumam na névoa, aumentando a percepção de profundidade. Neste teste os objetos também se movimentam aleatoriamente pela cena, assim como a distância máxima em que o *fog* será aplicado (Figura 3.2 a).

Para o último teste do nível 2, o teste 4, foram aplicados dois efeitos: iluminação/ sombreamento e o *fog* linear, simultaneamente. Desta maneira foi possível visualizar, com mais detalhes, o comportamento do sistema Android ao manipular objetos com múltiplos efeitos aplicados (Figura 3.2 b).



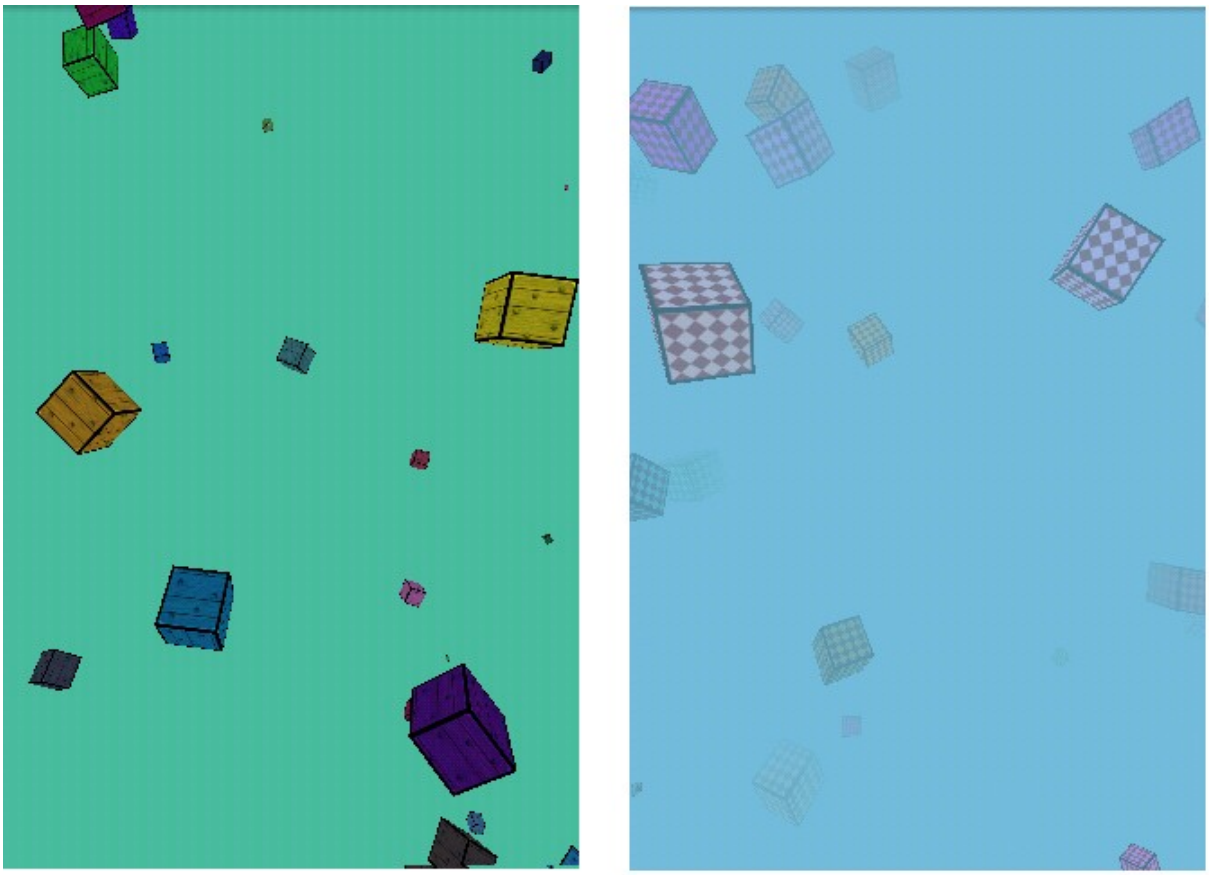
(a) (b)

Figura 3.2 – Telas capturadas durante a execução do teste 03 (a) e do teste 04 (b)

No teste 5 foi aplicada uma mesma textura aos objetos, sem nenhum outro tipo de efeito (Figura 3.3 a).

Para o teste 6, foram aplicados todos os efeitos vistos até aqui: iluminação/sombreamento, *fog* linear e textura. Todos eles tiveram seus parâmetros ajustados

aleatoriamente para que os cálculos fossem realizados a cada movimentação dos objetos pela cena, exigindo assim o processamento máximo do sistema testado (Figura 3.3 b).



(a) (b)

Figura 3.3 – Telas capturadas durante a execução do teste 05 (a) e do teste 06 (b)

Capítulo 4

Resultados

Neste capítulo serão apresentados os testes realizados e os resultados obtidos para cada uma das versões de Android analisadas, a v2.3.4 e a v4.0.4.

Para cada uma das versões do Android selecionadas foram executados os seis testes, com 10 repetições cada, totalizando 60 testes por versão. A aplicação desenvolvida permite configurar a duração e a quantidade de objetos utilizados em cada teste. Para a avaliação decidiu-se utilizar o tempo e o número de objetos fixos, variando de acordo com a versão em teste no momento. Os parâmetros podem ser alterados na tela inicial do aplicativo, como mostra a Figura 4.1.

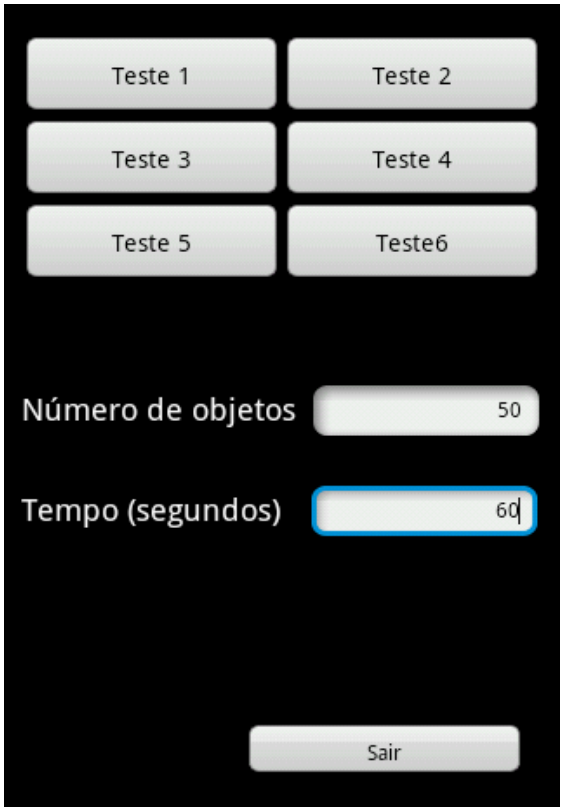


Figura 4.1 – Tela inicial do sistema

Os atributos dos objetos, como a cor e o tamanho, são aleatorizados em cada um dos objetos em cada um dos testes. O mesmo acontece para a cor de fundo da tela, a cor de cada uma das fontes de luz e suas posições. A avaliação de desempenho entre os diferentes testes e versões de Android foi realizada através da comparação de médias, priorizando o número de *fps*; para tanto foi exigido o máximo de processamento da CPU.

Após a execução de cada teste são exibidos os resultados, o total de *frames* que foram processados durante toda a execução da aplicação, o tempo total de execução, definida inicialmente, e o média de *fps*. Também é mostrado a porcentagem de uso de CPU utilizada ao executar o teste(Figura 4.2).

Figura 4.2 – Tela final de resultados

4.1 Android *Gingerbread* v2.3.4

Para os testes realizados no Android *Gingerbread* v2.3.4 foram utilizados 100 objetos em cena, sendo manipulados durante o tempo de 60 segundos. Foram escolhidos estes parâmetros para assegurar uma elevada utilização da capacidade de processamento da CPU, como mostram os resultados apresentados na Figura 4.3.

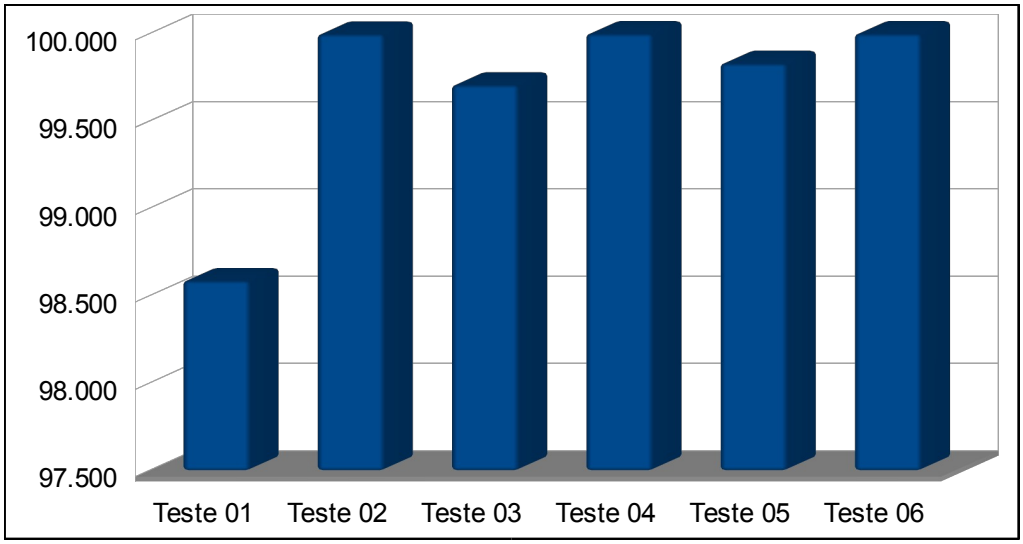


Figura 4.3 – Média de uso da CPU nos testes realizados com o Android *Gingerbread* v2.3.4

Analisando os dados referentes à média de *fps* utilizado em cada teste, Tabela 4.2, Figura 4.4 e Figura 4.5, é possível notar como a diferença na quantidade de *fps* entre alguns testes são bastante expressivas, como os testes 1 e 6. Essa diferença ocorre pelo fato de serem aplicados vários efeitos (iluminação/sombreamento, *fog* e textura) no teste 6 enquanto o teste 1 não se aplicou efeito gráfico.

Tabela 4.2 - Média de *fps* nos testes realizados com o Android *Gingerbread* v2.3.4

Amostra	Teste 01	Teste 02	Teste 03	Teste 04	Teste 05	Teste 06
1	54.33	43.15	51.37	41.22	49.32	39.42
2	52.92	42.68	51.62	41.00	49.47	39.15
3	53.60	42.75	51.23	41.33	49.78	38.83
4	52.33	42.08	51.58	41.18	49.07	39.22
5	53.12	43.05	51.53	41.67	49.48	38.97
6	51.48	42.13	51.73	40.90	49.80	39.25
7	52.77	42.53	51.37	42.28	49.53	38.85
8	53.38	42.37	50.92	42.28	49.38	38.13
9	52.92	42.60	51.22	41.73	49.85	38.92
10	52.98	42.35	51.50	41.48	49.58	38.88
Média	**52.983**	**42.569**	**51.407**	**41.507**	**49.526**	**38.962**

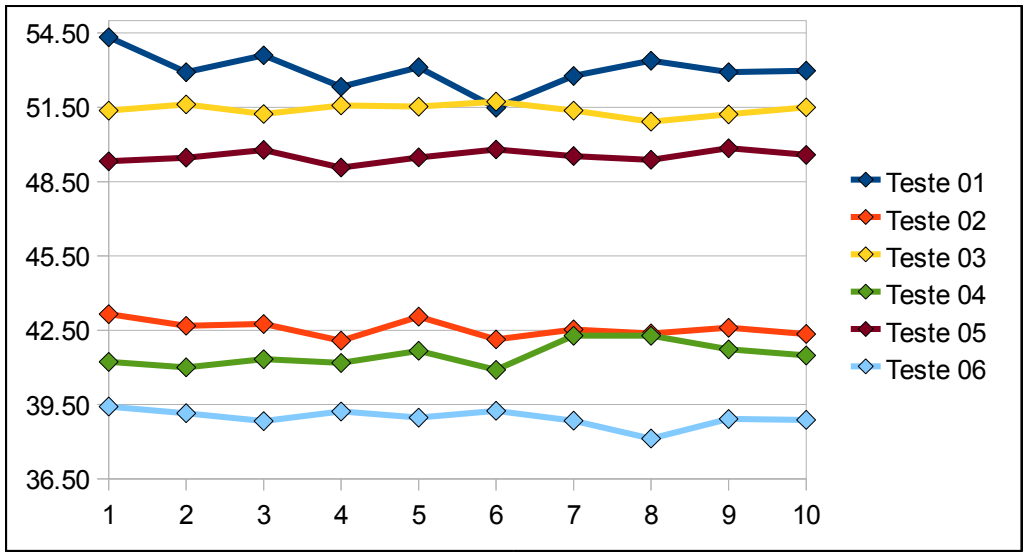


Figura 4.4 – Número de *fps* em cada teste realizado com o Android *Gingerbread* v2.3.4

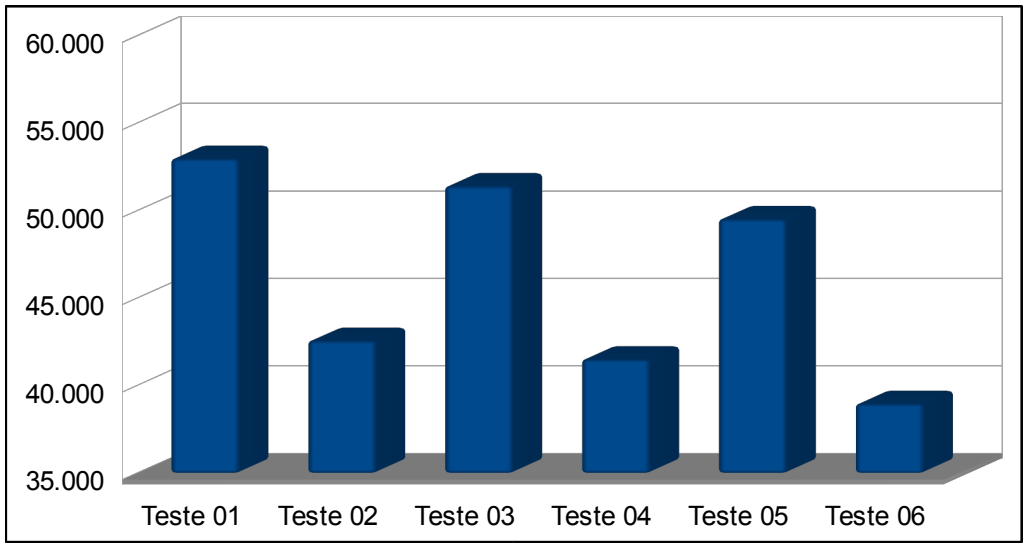


Figura 4.5 – Média de *fps* nos testes realizados com o Android *Gingerbread* v2.3.4

Ao comparar os dados referentes ao uso de CPU com os dados relativos ao número de *fps*, é visível que o número de *fps* está diretamente ligado à complexidade dos efeitos gráficos utilizado em cena que, por sua vez, está ligado diretamente ao uso de CPU.

Esta constatação também foi percebida nos testes realizados para ajuste de parâmetros. Utilizando um número de objetos menor, que não exija processamento máximo, é possível notar que os testes que aplicam os efeitos mais complexos utilizam mais processamento para

manter o mesmo nível de *fps* do que os testes com efeitos menos complexos, ou que não apliquem nenhum tipo de efeito gráfico.

Nestes dados é possível ver que todos os testes que aplicaram o efeito de iluminação/sombreamento, que foram os testes 2, 4 e 6, apresentaram menor número de *fps*. Isso se deve à quantidade de cálculos realizados para colorizar todos os *pixels* das faces dos objetos visíveis.

Ao compararmos os resultados do teste 2, que aplicado somente o efeito de iluminação/sombreamento, com o teste 3, que aplica somente o efeito de *fog*, é possível ver que o teste 2 tem média inferior ao teste 3. Isso porque o efeito *fog* seleciona apenas os objetos que realmente sofrerão algum tipo de manipulação pelos outros efeitos aplicados à cena, agindo como se fosse um algoritmo de recorte para objetos que estão fora dos limites, muito longe ou muito perto da câmera. Isso faz com que o número de cálculos sobre a cena diminua significativamente, aumentando o número de *fps*.

Esta análise permitiu verificar que o efeito de iluminação/sombreamento exige um nível de processamento maior do que o efeito de textura, sendo este um dos efeitos que mais utilizam processamento computacional durante a manipulação de objetos tridimensionais.

4.2 Android *Ice Cream Sandwich* v4.0.4

Para os testes realizados no Android *Ice Cream Sandwich* v4.0.4, foram utilizados os mesmos princípios dos testes realizados com a versão anterior; exigir processamento máximo de CPU para que seja possível analisar em detalhes a quantidade de *fps* produzidos nos testes.

Por esse motivo, os testes nesta versão foram executados sobre um número menor de objetos, apenas 10, pois com o mesmo número de objetos o desempenho desta versão se foi extremamente baixo, não passando dos 4 *fps*, mostrando que o processamento já estava sendo muito mais do que era suportado pelo aparelho.

Ao utilizar 10 objetos o processamento foi menor tornando-se suportado pelo aparelho e, mesmo assim, em todos os testes, foi utilizado 100% do processamento da CPU sem nenhum tipo de oscilação (Figura 4.6) fazendo com que o número de *fps* fosse mais aceitável.

Esse processamento elevado, mesmo com poucos objetos em cena, pode estar relacionado com o fato de que o Android v4.0.4 consome mais recursos de *hardware* que o Android v2.3.4, por conta de novas funcionalidades que exigem mais espaço de armazenamento e processamento para funcionamento do sistema.

Outro motivo que levou esta versão a um desempenho tão inferior pode ser o fato do *CyanogenMod* 9 não ser uma versão original e também por não ser uma versão desenvolvida para o aparelho em que os testes foram realizados, podendo ter afetado o desempenho de algum *hardware* utilizado para processamento gráfico.

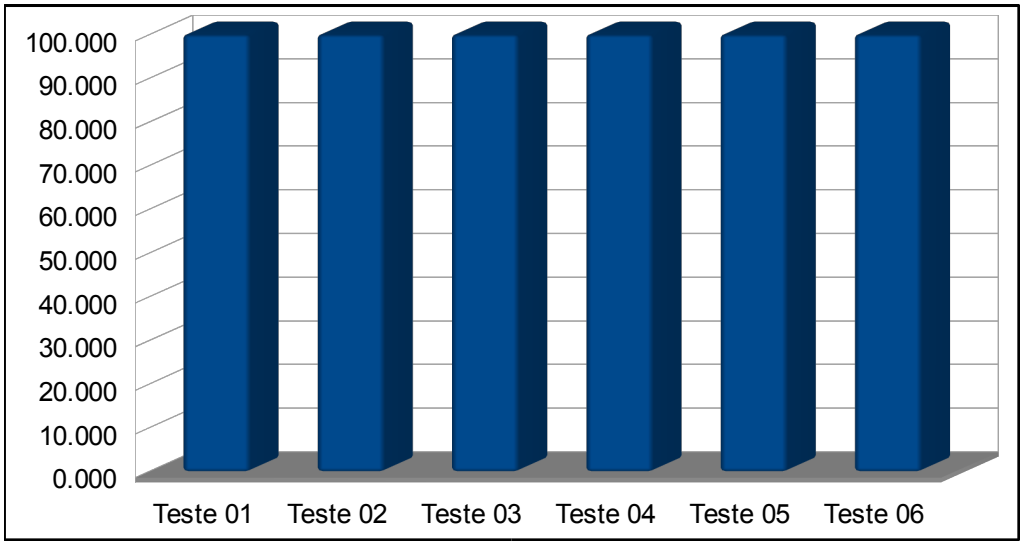


Figura 4.6 – Média de uso da CPU nos testes realizados com o Android *Ice Cream Sandwich* v4.0.4

Apesar do número de objetos ser menor, o comportamento dos testes em relação ao número de *fps* continuou o mesmo, quando comparados aos testes realizados com a versão 2.3.4, como podem ser vistos na Tabela 4.3 e nas Figuras 4.7 e 4.8.

Os testes 2, 4 e 6, que aplicam o efeito de iluminação/sombreamento foram os que obtiveram menor média de *fps* enquanto o teste 1 foi o que teve a maior média de *fps* tendo a menor quantidade de processamento, como nos testes realizados com a versão 2.3.4.

Tabela 4.3 – Número de *fps* obtidos nos testes realizados com o Android *Ice Cream Sandwich* v4.0.4

Teste 01	Teste 02	Teste 03	Teste 04	Teste 05	Teste 06
38.02	36.87	38.00	36.80	38.27	36.70
38.15	36.85	38.00	36.78	38.05	36.68
36.78	36.87	37.95	36.75	37.98	36.63
38.18	36.87	38.05	36.73	38.03	36.70
38.18	36.75	37.97	36.85	38.12	36.67
38.17	36.90	38.00	36.72	37.98	36.68
38.20	36.88	37.97	36.78	38.08	36.82
38.15	36.87	37.95	36.75	37.87	36.88
38.07	36.88	38.05	36.83	37.98	36.87
38.20	36.80	38.00	36.88	38.08	36.77
38.010	**36.854**	**37.994**	**36.787**	**38.044**	**36.740**

Figura 4.7 – Número de *fps* em cada teste realizado com o Android *Ice Cream Sandwich* v4.0.4

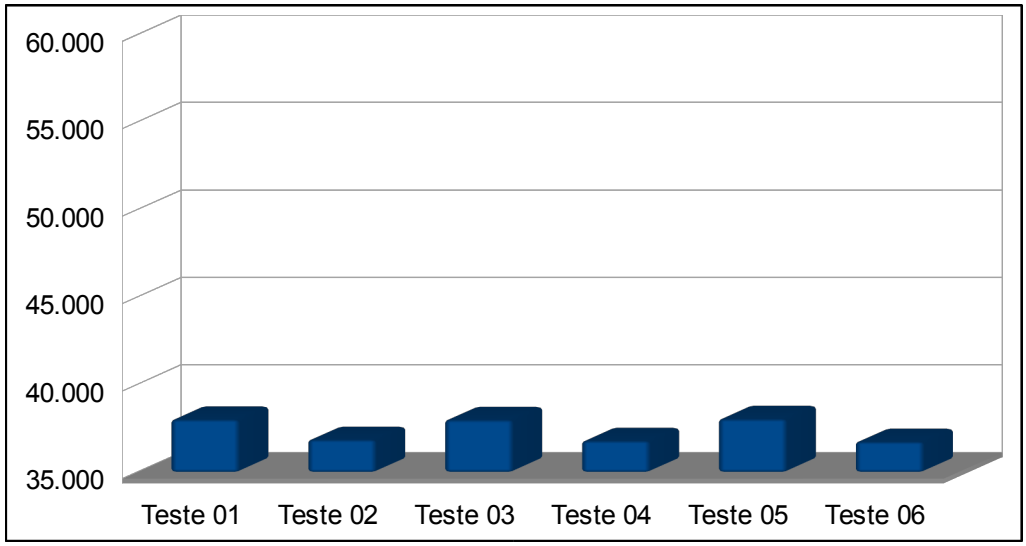


Figura 4.8 – Média de *fps* nos testes realizados com o Android *Ice Cream Sandwich* v4.0.4

Capítulo 5

Conclusão

Há pouco tempo surgiu um novo sistema operacional para *smartphones*, o Android, que se tornou um dos mais utilizados em apenas cinco anos.

Por ter se expandido rapidamente e por possuir várias atualizações em curtos períodos de tempo, a evolução do sistema Android fez com que surgissem alguns problemas. Um deles é a rápida obsolescência dos *smartphones*, fazendo com que se tornem tecnologicamente ultrapassados, pois faltam atualizações nas versões de Android executados nestes aparelhos.

Por existirem aparelhos com *hardwares* diversos no mercado, muitas funcionalidades do sistema Android podem ser comprometidas como, por exemplo, a capacidade de executar aplicações tridimensionais.

Visando avaliar a compatibilidade e o desempenho de aplicações gráficas entre diferentes versões do Android, foram realizados diversos testes com as versões *Gingerbread* v2.3.4 e *Ice Cream Sandwich* v4.0.4 em um mesmo *smartphone*.

A aplicação foi feita com base no OpenGL ES 1.0, utilizando *pipeline* fixo, pois além de tornar mais fácil o desenvolvimento da aplicação, as versões superiores mais recentes da API poderiam trazer problemas de compatibilidade devido ao uso do *pipeline* programável, tornando a aplicação incompatível entre as versões utilizadas.

Como os testes não apresentaram problemas de compatibilidade nas duas versões avaliadas restou analisar o fator desempenho.

Os testes foram realizados com um número de objeto que fizessem com que o uso da CPU fosse o máximo, tornando a diferença de *fps* entre as versões um parâmetro de avaliação do desempenho

Analisando os resultados dos testes descritos no capítulo 4, pôde-se observar que as duas versões mantiveram um comportamento semelhante. Os testes que realizaram maior processamento (2, 4 e 6) são os que aplicaram o efeito de iluminação/sombreamento

possuindo uma média de *fps* menor que os testes 1, 3 e 5, que realizaram menos processamento. Isso mostra que a iluminação/sombreamento gera mais processamento em uma cena do que a aplicação de textura nos objetos ou a aplicação do efeito *fog* na cena. Mas a principal diferença nos testes entre versões está na quantidade de objetos em cena, que foi menor na *Ice Cream Sandwich* v4.0.4.

Apesar do número de objetos em cena ser 10 vezes menor, a *Ice Cream Sandwich* v4.0.4 não chegou, em nenhum momento, próxima dos resultados obtidos com a *Gingerbread* v2.3.4, mostrando que o seu desempenho foi significativamente menor. Isso tem muita influência do processamento realizado por cada versão apenas para mantê-la funcionando; como a versão 4.0.4 possui novas funcionalidades e que exigem mais processamento e memória, é compreensível que seu desempenho gráfico seja reduzido.

Ao realizar os testes na versão 4.0.4 com o mesmo número de objetos dos testes da outra versão (100 objetos), a diferença de desempenho se tornou ainda maior. Enquanto a versão 2.3.4 manteve médias superiores a 38 *fps*, chegando a 52 *fps* nos testes mais simples, os testes na versão 4.0.4 não passaram de 4 *fps*.

Mesmo após realizar vários testes e constatar que a versão 4.0.4 exige mais processamento em funções básicas do sistema, não foi possível chegar a uma conclusão precisa sobre o real motivo da diferença de desempenho entre as duas versões. Acredita-se que o principal motivo seja o fato da *custom* ROM utilizada (*CyanogenMod* 9) não ser própria para o aparelho; com isso, alguma funcionalidade ou *hardware* utilizado pelo sistema Android, como o processador gráfico, pode ter sido subutilizado, afetando assim o desempenho gráfico do aparelho.

A implementação de um aplicativo gráfico, como o desenvolvido neste trabalho, pode ser considerado fácil para pessoas com alguma experiência em programação para Android e openGL. Porém, para pessoas que nunca desenvolveram aplicativos para Android ou nunca utilizaram a API openGL, haverão dificuldades durante a implementação. Recomenda-se, inicialmente, entender como funciona um aplicativo para Android, pois isso evitará alguns problemas como, por exemplo, o gerenciamento de *activities* durante a execução. Conhecendo como uma *activity* se comporta é possível manipulá-la de maneira adequada, finalizando-a quando necessário ou fazendo algum tipo de processamento quando alguma delas entrar no estado *pause*.

A maior dificuldade encontrada no desenvolvimento deste trabalho foi conseguir executar corretamente a aplicação no celular, pois inicialmente não se dispunha do aparelho, realizando os testes apenas em emulador. Ao migrar o aplicativo para o *smartphone* surgiram erros que não aconteciam no emulador e, com isso, a aplicação deixava de executar corretamente. No emulador o *hardware* utilizado é o de um computador que, em comparação com o *smartphone*, possui recursos praticamente ilimitados.

Este tipo de erro ocorreu pela falta de experiência no desenvolvimento de aplicações para a plataforma Android, juntamente com a falta de experiência na programação utilizando o OpenGL ES. O erro foi corrigido através da reescrita e reestruturação do código, testando a aplicação diretamente no aparelho.

5.1 Trabalhos futuros

Como trabalhos futuros pode-se utilizar uma nova abordagem para avaliar quais motivos levaram a versão 4.0.4 a ter um desempenho tão inferior a versão 2.3.4. Entre os elementos suspeitos podemos citar subutilização do *hardware* pela *custom* ROM. Esta avaliação pode ser feita com um aparelho que possua duas, ou mais, versões oficiais do sistema Android.

Para que os resultados se tornem mais precisos, o aplicativo pode ser melhorado acrescentando mais efeitos aos testes e mais requisitos para as comparações, como a quantidade de memória RAM utilizada em cada teste e a qualidade das imagens geradas.

APÊNDICE A – PREPARANDO O AMBIENTE DE DESENVOLVIMENTO ANDROID [16]

A.1 Obtendo as ferramentas necessárias

Para iniciar o desenvolvimento de um aplicativo Android é necessário instalar um ambiente de desenvolvimento; o mais recomendado é o Eclipse, mas também há a opção do NetBeans. O Eclipse é o mais recomendado, pois é uma plataforma que suporta várias linguagens de desenvolvimento e também por ser expansível através de *plugins*, o que facilita a adição de novas ferramentas para o desenvolvimento de sistemas.

O Eclipse pode ser encontrado no link www.eclipse.org/downloads. Ele está disponível em seis versões: Windows (32 e 64 bits), Mac OS X (Cacau 32 e 64) e Linux (32 e 64 bits). Neste exemplo foi utilizado o Eclipse para Windows versão 32 bits.

O próximo passo é instalar o Android SDK, que pode ser encontrado no link http://developer.android.com/sdk/. Esse SDK contém um depurador, bibliotecas, um emulador, códigos exemplo e tutoriais. Após fazer o *download* descompacte o conteúdo do arquivo para a pasta onde foi instalado o Eclipse.

Após a instalação do Eclipse e do Android SDK é preciso instalar o ADT, um plugin para o Eclipse que suporta a criação e depuração de aplicações Android. Através dele é possível:

- Criar novos projetos Android;
- Acessar ferramentas que darão acesso à emuladores e dispositivos Android;
- Compilar e depurar aplicações Android;
- Fazer pedidos de exportação dos aplicativos Android para pacotes Android (APK);
- Criar certificados digitais para assinar os pacotes.

Para instalar o ADT, primeiro executa-se o arquivo eclipse.exe, localizado na pasta do Eclipse. Quando o Eclipse for iniciado será solicitado o nome da pasta que servirá como espaço de trabalho; nesta pasta serão salvos todos os projetos criados pelo Eclipse.

Com o Eclipse instalado e funcionando, selecione a opção *Help* e em seguida clique em *Install New Software* (Figura A.1). Na janela de instalação digite http://dl-ssl.google.com/android/eclipse na caixa de texto (Figura A.2) e clique em *Add*.

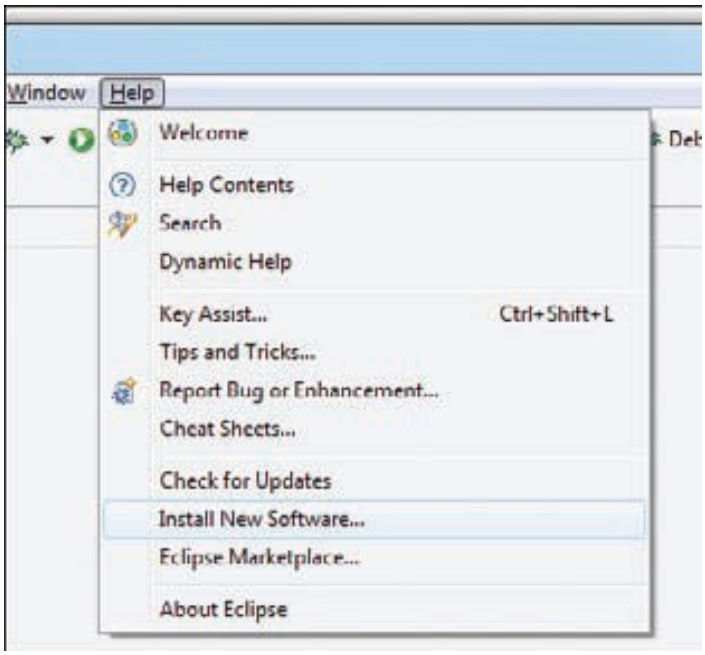


Figura A.1

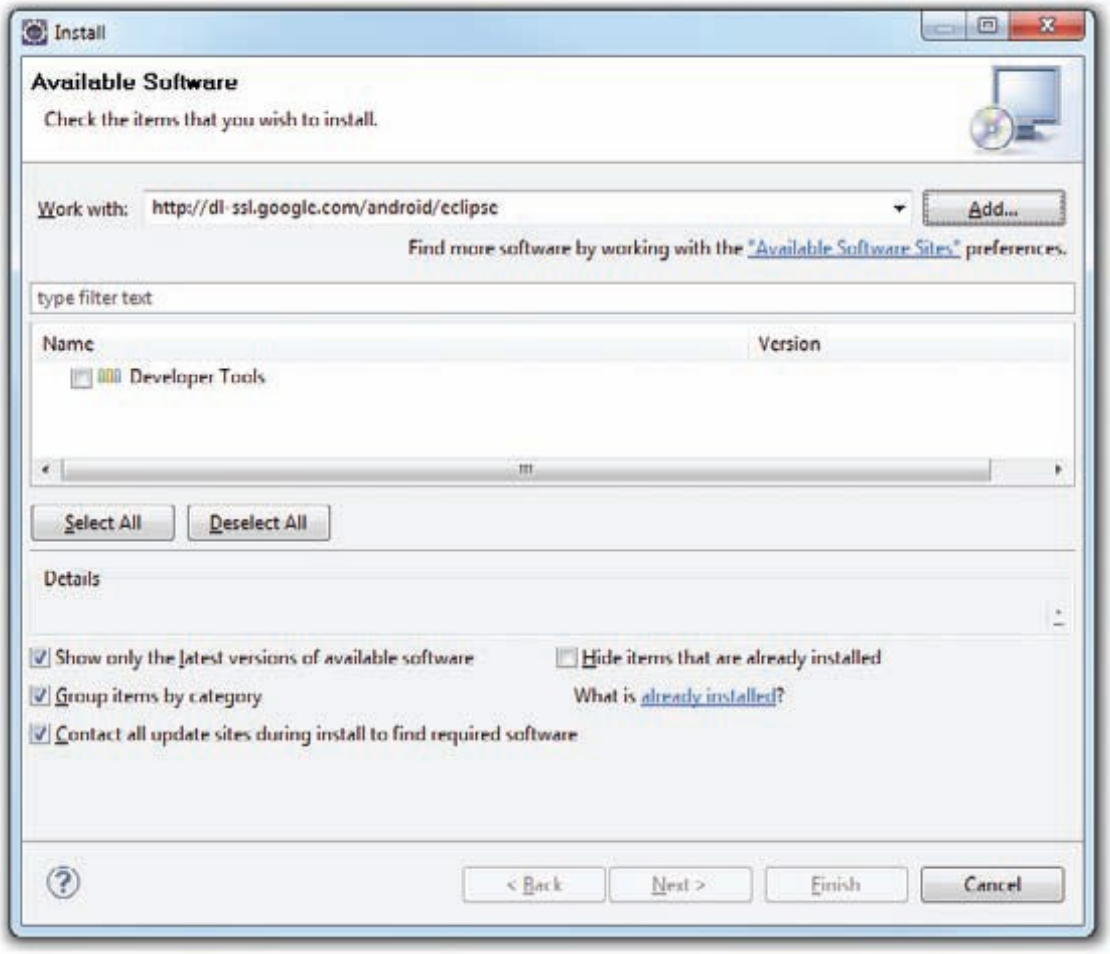


Figura A.2

Ao expandir o item *Developer Tools* deverá aparecer o seu conteúdo: *Android DDMS, Android Development Tools e Android Hierarchy Viewer*. Marque as três caixas e clique em *Next* (Figura A.3). Ao aparecer a janela de detalhes de instalação, clique em *Next*. Após será solicitada a revisão de licença para as ferramentas, marque a opção para aceitar os termos de licença e depois em *Finish*.

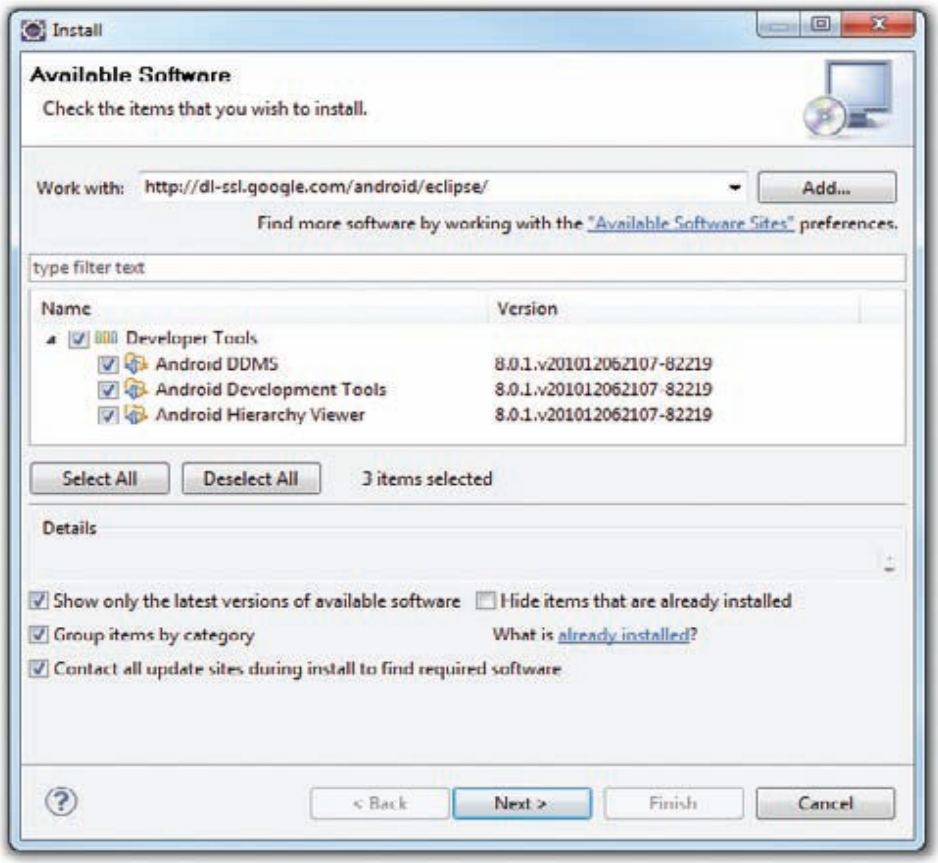


Figura A.3

Concluídas estas etapas, o Eclipse avançará para o *download* das ferramentas e as instalará; esta etapa pode levar algum tempo.

Após a instalação do ADT será solicitado o reinício do Eclipse. Após reiniciar o Eclipse clique no menu *Window* e depois em *Preferences* (Figura A.4). Na janela que aparecerá selecione Android, no canto esquerdo. Feito isso surgirá uma mensagem de erro informando que o SDK não foi criado.

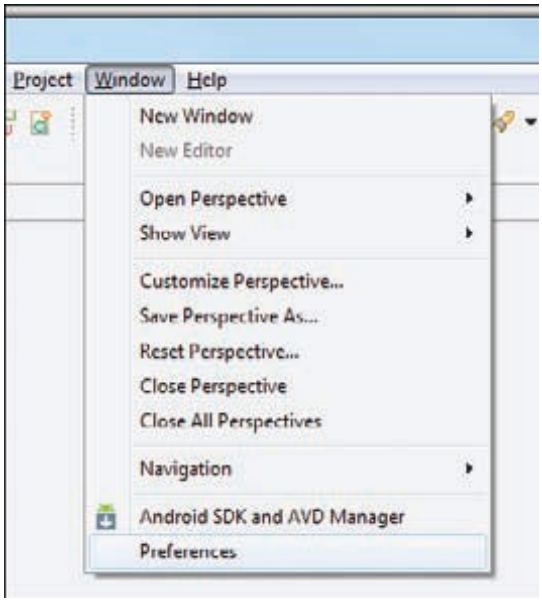


Figura A.4

No campo de texto referente à localização do SDK, será informado o local onde o SDK está instalado. Este passo será mostrado na próxima seção.

A.2 Criando os AVDs (Android Virtual Devices)

O AVD é um dispositivo que emula um sistema Android para um dispositivo real. Esse emulador é bastante utilizado para a realização de testes em aplicativos em desenvolvimento.

É possível criar vários AVDs para testar uma aplicação em várias configurações diferentes. Testes deste tipo são importantes para se conhecer o comportamento que a aplicação terá ao ser executada em diferentes dispositivos Android.

Para criar um AVD basta clicar no menu *Window* e depois em *Android SDK and AVD Manager*. Após ser aberta a janela de gerenciamento selecione a opção *Available packages* no painel esquerdo (Figura A.5), mostrando os vários pacotes disponíveis para a criação de AVDs para emular as diferentes versões do Android.

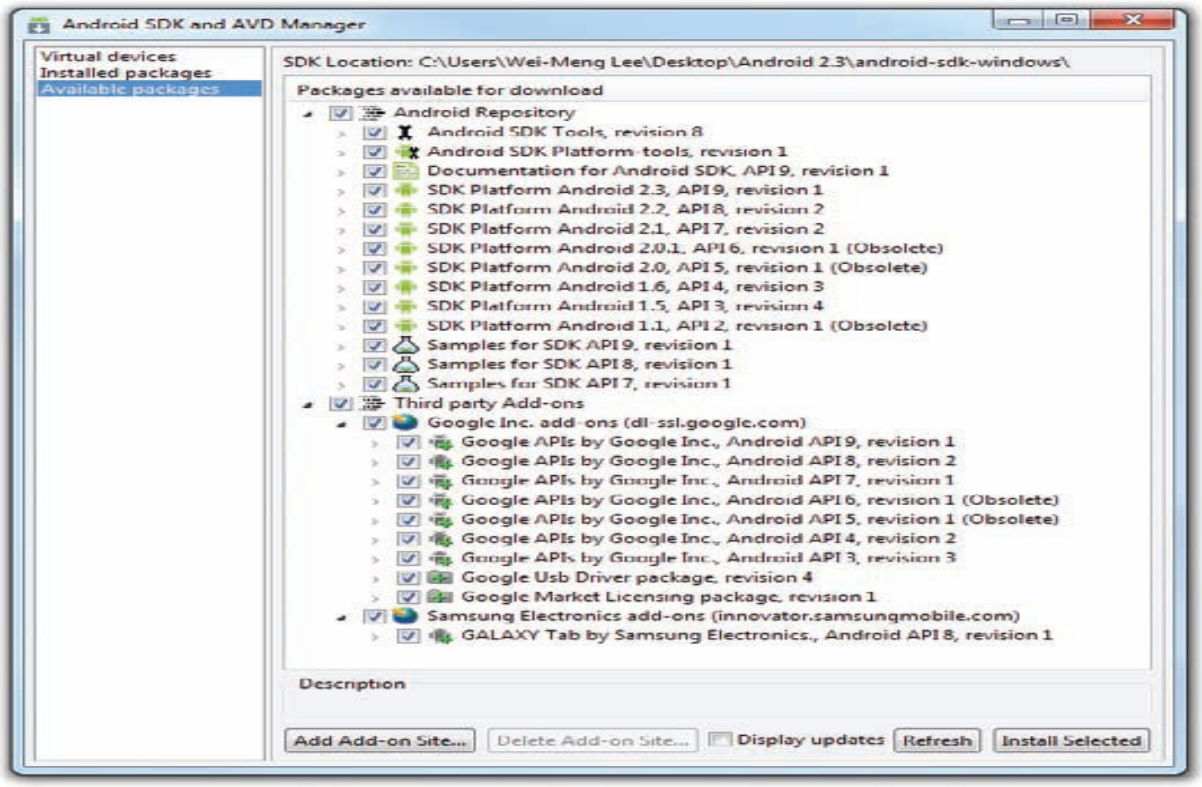


Figura A.5

Após verificar quais pacotes serão necessários para a aplicação, selecione-os e em seguida clique no botão *Install Selected*. Este processo pode demorar algum tempo; por isso é recomendável que se instale somente os pacotes necessários para a aplicação e depois, quando tiver mais tempo, instalar os demais pacotes.

Cada versão do Android é identificada pelo nível de API como, por exemplo, o Android 2.2 que possui a API nível 8 e o Android 2.3 que possui a API nível 9. Em cada nível estão disponíveis dois tipos de plataformas; no caso do Android 2.3 (API 9) é oferecida a plataforma SDK Android 2.3 e as APIs do Google pelo Google Inc; a principal diferença é que a plataforma API do Google contém as bibliotecas do Google Maps, portanto essa plataforma só será utilizada caso a aplicação necessitar do Google Maps, como aplicativos GPS.

Clicando no item *Virtual Devices* localizado no painel esquerdo da janela e em seguida em *New*, localizado no painel direito, abre-se uma janela onde será possível criar um AVD (Figura A.6). Nesta janela basta informar um nome para o emulador e qual a versão do Android que será emulada. Após clique no botão *Create AVD*.

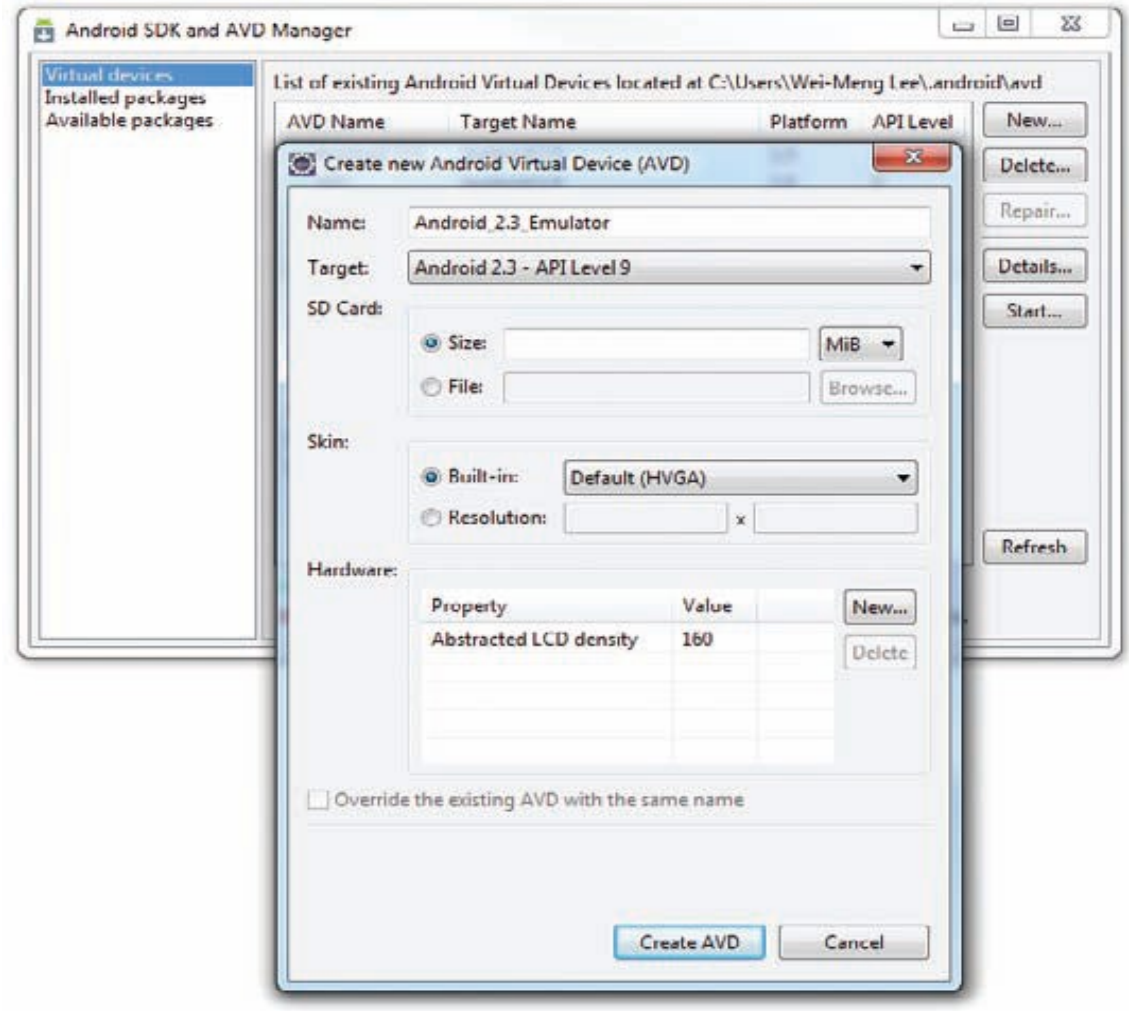


Figura A.6

Após estes passos criou-se um emulador Android e tudo estará pronto para os testes dos aplicativos criados. É aconselhável criar vários emuladores de versões diferentes para que seja possível testar a aplicação em diferentes dispositivos.

A.3 Criando o primeiro aplicativo Android

Após fazer o *download* e a instalação das ferramentas necessárias pode-se iniciar a construção da primeira aplicação para Android. Neste exemplo será criada a aplicação "*Hello World*".

- Passo 1: Crie um novo projeto no Eclipse, selecionando *File – New – Project*.
- Passo 2: Expandir a pasta Android e selecionar *Android Project* (Figura A.7)

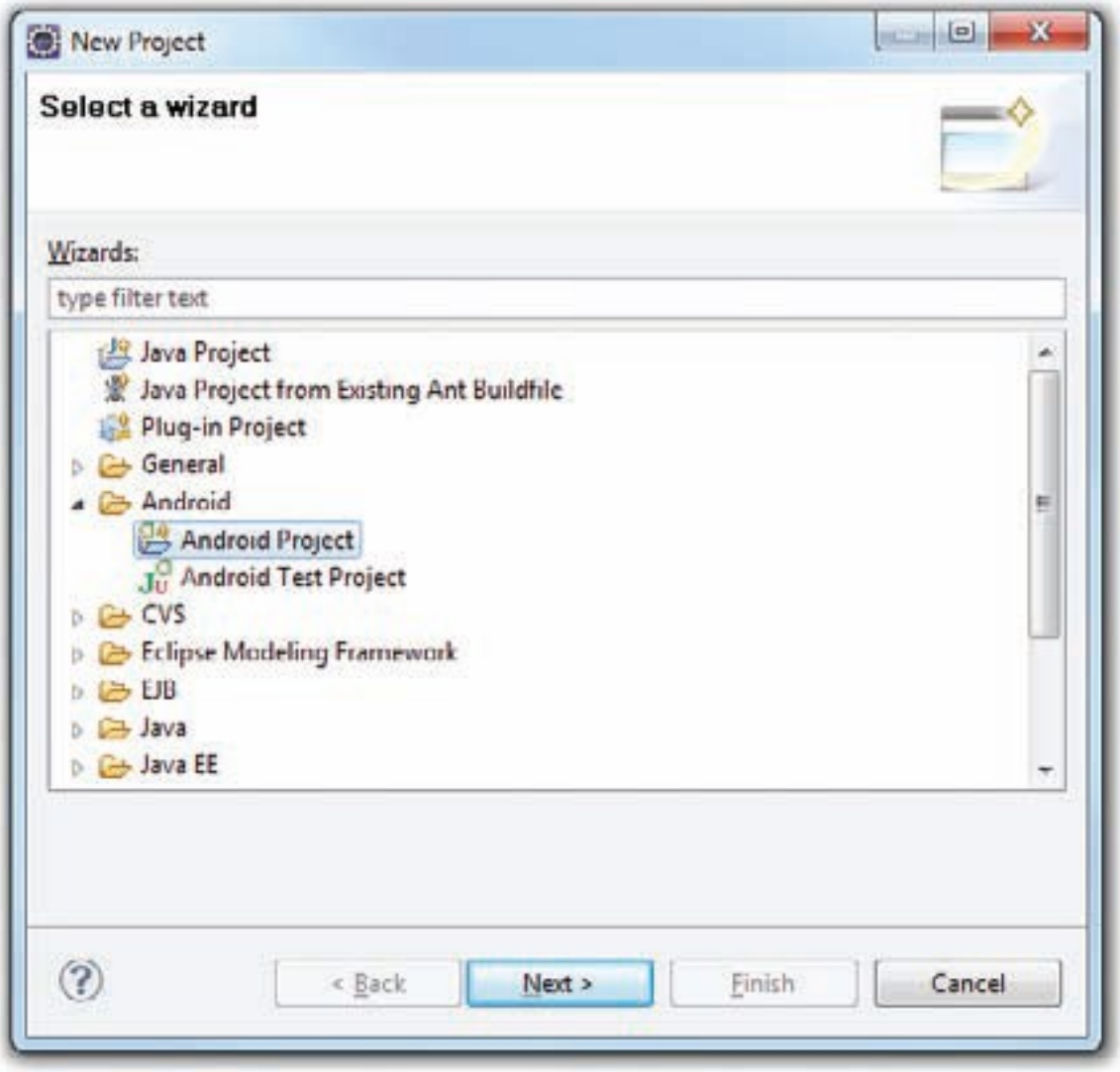


Figura A.7

- Passo 3: Nomear o projeto, nomear a aplicação, nomear o pacote e criar uma classe Activity, selecionando a versão do Android a ser utilizada (Figura A.8). O nome do pacote deve conter, obrigatoriamente, ao menos um caractere ponto (.). Por convenção recomenda-se usar o nome do domínio em ordem inversa seguido pelo nome do projeto.

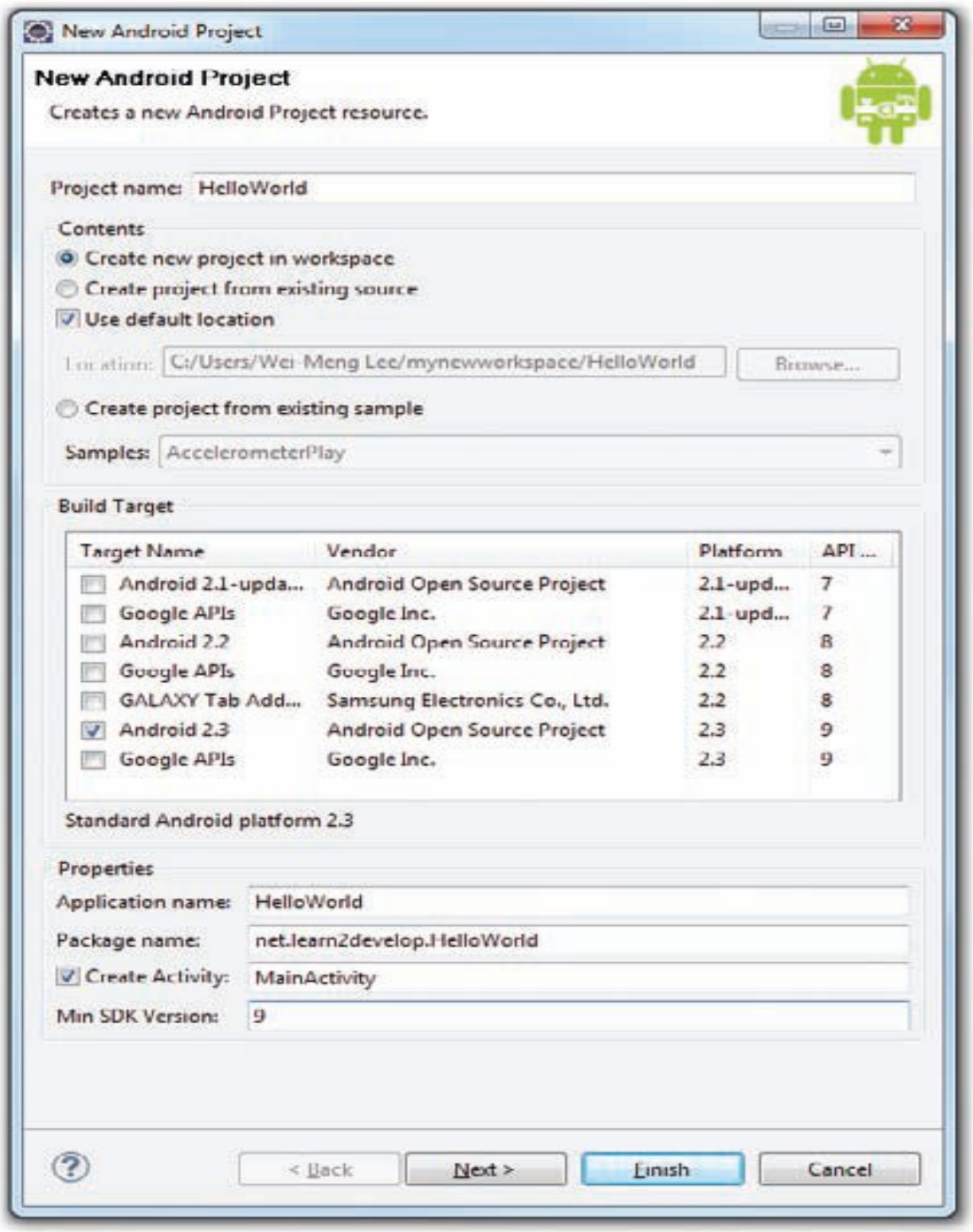


Figura A.8

- Passo 4: Após criar o projeto, no painel esquerdo (*Package Explorer*), clique nas setas exibindo cada um dos itens do projeto. Na pasta *res/layout* abra o arquivo *main.xml* (Figura A.9).

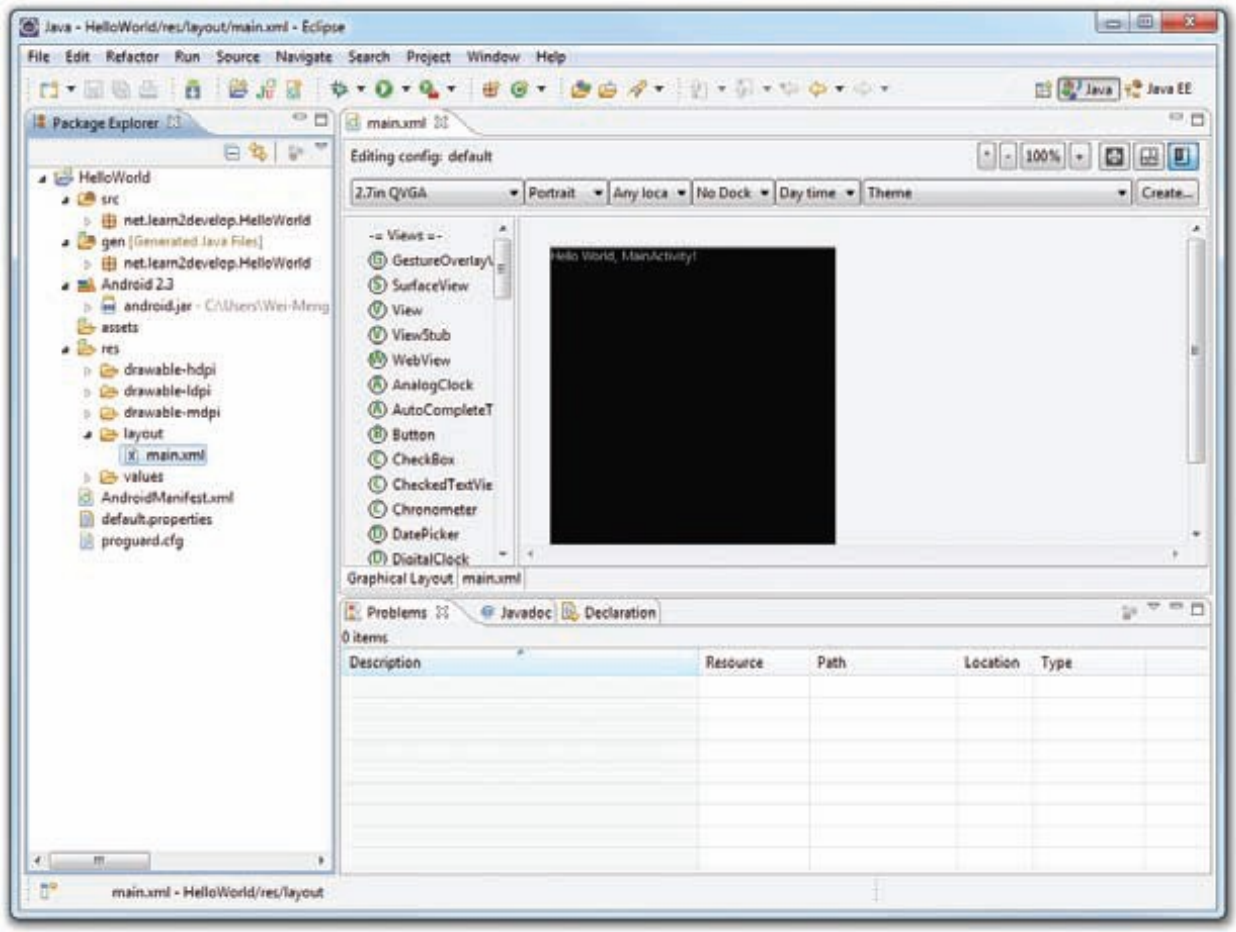


Figura A.9

- Passo 5: O arquivo .xml define o leiaute do aplicativo a ser criado. Por padrão o modo de visão é o leiaute, uma visão gráfica da tela. Para modificar o modo de visão clique guia *main.xml* na parte inferior da janela (Figura A.10).

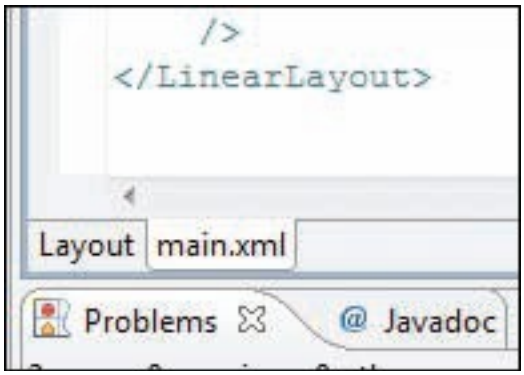


Figura A.10

- Passo 6: Adicione o código abaixo e salve-o, pressionando *Ctrl+S*. Neste código o arquivo *main.xml* foi formatado para exibir a frase "Este é o meu primeiro aplicativo para Android!" em uma textView. Também foi criado um botão com o texto "Botão Teste!". Este arquivo contém a interface inicial do aplicativo, exibida quando a Activity principal é carregada:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/hello" />
        <TextView
                android:id="@+id/textView1"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Este é o meu primeiro aplicativo para Android"
                android:textAppearance="?android:attr/textAppearanceMedium" />
        <Button
                android:id="@+id/button1"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="Botao Teste" />
</LinearLayout>
```

- Passo 7: Execute o teste no emulador Android, selecionando o nome do projeto e teclando F11. Uma janela para selecionar o modo de depuração do aplicativo será exibida. Selecione a opção *Android Application* (Figura A.11) e clique em OK.

Observação: Algumas instalações do Eclipse possuem um erro. Após cria um projeto é exibido um erro ao tentar depurar o aplicativo. Isso ocorre mesmo quando não existe nenhum erro, e até mesmo quando não houver nenhuma modificação no código inicial. Para resolver este problema, basta deletar o arquivo R.java, localizado em gen/net/<nome do

pacote criado>. O Eclipse criará este arquivo automaticamente ao depurar o aplicativo novamente. Feito isso, o projeto será executado corretamente.

Figura A.11

- Passo 8: O emulador será iniciado e executará o aplicativo (Figura A.12). Quando o aplicativo é depurado no Emulador Android, ele é instalado automaticamente no Emulador, e com isso não será preciso depurar pelo Eclipse novamente caso queira executar o aplicativo, a não ser que alguma modificação no código tenha ocorrido.

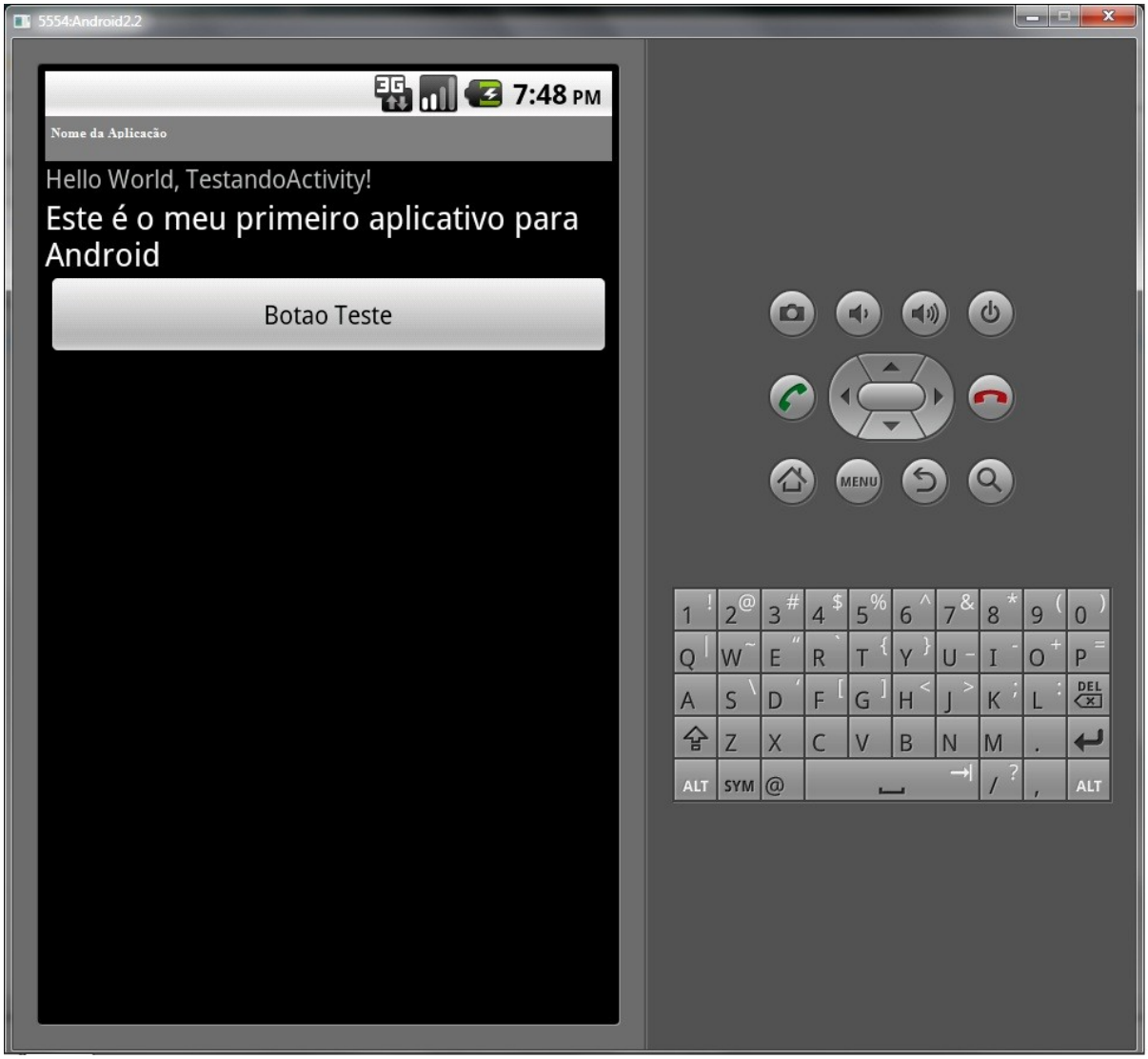


Figura A.12

A.4 Conhecendo os arquivos do projeto

Conhecer os arquivos que fazem parte de um projeto Android ajuda a entender como desenvolver aplicativos para Android, evitando que alguns erros venham a ser cometidos. Os arquivos de um projeto Android estão agrupados nas seguintes pastas:

- src – Contém os arquivos de origem Java para o seu projeto. Por exemplo, o arquivo *Activity* principal criado na aplicação anterior. É nele que o código da aplicação é escrito.
- Android 2.3 *library* – Este item contém um arquivo, android.jar, que contém todas as bibliotecas de classe necessárias para uma aplicação Android.

- gen – Contém o arquivo R.java, um arquivo gerado pelo compilador que faz referência a todos os recursos encontrados em seu projeto. Este arquivo não deve ser modificado.
- *assets* – Esta pasta contém todos os *assets* utilizados pela sua aplicação, tais como HTML, arquivos de texto, bancos de dados, etc.
- res – Esta pasta contém todos os recursos utilizados na sua aplicação. Ele também contém algumas outras subpastas: *drawable-<resolution>*, *layout* e *values*.
- AndroidManifest.xml - Este é o arquivo de manifesto para a sua aplicação Android. Aqui você pode especificar as permissões necessárias para a sua aplicação, bem como outras características (tais como a intenção de filtros, receptores, etc.)

APÊNDICE B – CÓDIGO FONTE DO PROJETO

Neste apêndice é apresentado todo o código fonte do projeto desenvolvido durante este trabalho, afim de mostrar como desenvolver uma aplicação gráfica básica para Android.

B.1 Layouts

Um *layout* define a estrutura visual para a interface de usuário, contendo todos os componentes de cada interface. A seguir são apresentadas os três *layouts* que foram utilizados no sistema.

B.1.1 Main.xml

Layout inicial da aplicação. Nele contém os campos referentes ao número de objetos a ser usado e o tempo de execução da aplicação. Ele possui também seis botões que fazem referência a cada um dos testes.

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/
              apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <AbsoluteLayout
        android:id="@+id/AbsoluteLayout1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >

        <Button
            android:id="@+id/botaoTeste1"
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_x="10dp"
            android:layout_y="20dp"
            android:text="Teste 1" />

        <Button
            android:id="@+id/botaoTeste2"
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_x="160dp"
            android:layout_y="20dp"
```

```
    android:text="Teste 2" />

<Button
    android:id="@+id/botaoTeste3"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="10dp"
    android:layout_y="70dp"
    android:text="Teste 3" />

<Button
    android:id="@+id/botaoTeste4"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="160dp"
    android:layout_y="70dp"
    android:text="Teste 4" />

<Button
    android:id="@+id/botaoTeste5"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="10dp"
    android:layout_y="120dp"
    android:text="Teste 5" />

<Button
    android:id="@+id/botaoTeste6"
    android:layout_width="150dp"
    android:layout_height="wrap_content"
    android:layout_x="160dp"
    android:layout_y="120dp"
    android:text="Teste6" />

<TextView
    android:id="@+id/tvNumber"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="10dp"
    android:layout_y="230dp"
    android:text="Número de objetos"
    android:textAppearance= "?android:attr/
      textAppearanceMedium"/>

<EditText
    android:id="@+id/etNumber"
    android:layout_width="130dp"
    android:layout_height="35dp"
    android:layout_x="178dp"
    android:layout_y="227dp"
    android:gravity="right"
    android:inputType="number"
    android:text="300"
    android:textSize="12dp" />

<TextView
    android:id="@+id/tvTempo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="10dp"
    android:layout_y="290dp"
    android:text="Tempo (segundos)"
    android:textAppearance="?android:attr/
      textAppearanceMedium"/>

<EditText
    android:id="@+id/etTempo"
```

```
            android:layout_width="130dp"
            android:layout_height="35dp"
            android:layout_x="177dp"
            android:layout_y="287dp"
            android:gravity="right"
            android:inputType="number"
            android:text="10"
            android:textSize="12dp" />

        <Button
            android:id="@+id/botaoSair"
            android:layout_width="162dp"
            android:layout_height="33dp"
            android:layout_x="138dp"
            android:layout_y="431dp"
            android:text="Sair"
            android:textSize="12dp" />
    </AbsoluteLayout>

</RadioGroup>
```

B.1.2 Execucao.xml

Layout referente à execução do sistema, não possui nenhum componente, pois é apenas um *layout* para execução.

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/
              apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >

        <LinearLayout
            android:id="@+id/linearLayout2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" >
        </LinearLayout>

        <FrameLayout
            android:id="@+id/frame"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" >
        </FrameLayout>
    </LinearLayout>

</RadioGroup>
```

B.1.3 Resultados.xml

Layout final da aplicação, possuindo apenas os campos para a exibição dos resultados.

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

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >

<AbsoluteLayout
    android:id="@+id/absoluteLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <TextView
        android:id="@+id/tvDadosFPS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="70dp"
        android:layout_y="20dp"
        android:text="Frames por segundo"
        android:textAppearance="?android:attr/
          textAppearanceMedium"/>

    <TextView
        android:id="@+id/tvTotalFrames"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="85dp"
        android:layout_y="55dp"
        android:text="Total de frames:"
        android:textAppearance="?android:attr/
          textAppearanceSmall"/>

    <TextView
        android:id="@+id/tvTempoTotal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="85dp"
        android:layout_y="90dp"
        android:text="Tempo total(s):"
        android:textAppearance="?android:attr/
          textAppearanceSmall"/>

    <TextView
        android:id="@+id/tvFPS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="85dp"
        android:layout_y="125dp"
        android:text="Número de FPS:"
        android:textAppearance="?android:attr/
          textAppearanceSmall"/>

    <TextView
        android:id="@+id/tvResultTotalFrames"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="190dp"
        android:layout_y="55dp"
        android:text="0"
        android:textAppearance="?android:attr/
          textAppearanceSmall"/>

    <TextView
        android:id="@+id/tvResultTempoTotal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="190dp"
        android:layout_y="90dp"
        android:text="0"
        android:textAppearance="?android:attr/
```

```
            textAppearanceSmall"/>

        <TextView
            android:id="@+id/tvResultFPS"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="190dp"
            android:layout_y="125dp"
            android:text="0.00"
            android:textAppearance="?android:attr/
              textAppearanceSmall"/>

        <TextView
            android:id="@+id/tvDadosCPU"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="110dp"
            android:layout_y="195dp"
            android:text="Uso da CPU"
            android:textAppearance="?android:attr/
              textAppearanceMedium"/>

        <TextView
            android:id="@+id/tvPorcUtilizado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="105dp"
            android:layout_y="230dp"
            android:text="Pocentagem:"
            android:textAppearance="?android:attr/
              textAppearanceSmall"/>

        <TextView
            android:id="@+id/tvResultPorcDeCPU"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="190dp"
            android:layout_y="230dp"
            android:text="0.00"
            android:textAppearance="?android:attr/
              textAppearanceSmall"/>
    </AbsoluteLayout>

</LinearLayout>
```

B.2 Activities

Uma *activity* é uma classe gerenciadora de interfaces da aplicação, fazendo toda a parte de interação com o usuário, semelhante a um *JFrame* ou um *JPanel* utilizados em uma aplicação para *desktop*. A seguir é apresentado as três *activitys* utilizadas na aplicação deste trabalho, sendo que cada uma delas está relacionada a cada um dos *Layouts* descritos anteriormente.

B.2.1 ActivityInicial

Activity inicial da aplicação Nela é definido o número de objetos em cena, o tempo de execução e qual dos seis testes será executado.

```
package com.TccNehe;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.EditText;

public class ActivityInicial extends Activity {

	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		// Cria uma aplicaçao fullscream
		this.requestWindowFeature(Window.FEATURE_NO_TITLE);
		getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
				WindowManager.LayoutParams.FLAG_FULLSCREEN);
		// ----------------------------------

		setContentView(R.layout.main);

		Button botaoTeste1 = (Button) findViewById(R.id.botaoTeste1);
		Button botaoTeste2 = (Button) findViewById(R.id.botaoTeste2);
		Button botaoTeste3 = (Button) findViewById(R.id.botaoTeste3);
		Button botaoTeste4 = (Button) findViewById(R.id.botaoTeste4);
		Button botaoTeste5 = (Button) findViewById(R.id.botaoTeste5);
		Button botaoTeste6 = (Button) findViewById(R.id.botaoTeste6);

		Button botaoSair = (Button) findViewById(R.id.botaoSair);

		final EditText numObject = (EditText) findViewById(R.id.etNumber);
		final EditText tempoTeste = (EditText) findViewById(R.id.etTempo);

		numObject.setText("" + VariaveisControle.NUM_OBJECTS);
		tempoTeste.setText("" + VariaveisControle.TIME_TOTAL / 1000);

		botaoTeste1.setOnClickListener(new View.OnClickListener() {
			public void onClick(View view) {
				VariaveisControle.contFechar++;

				VariaveisControle.TIPO_TESTE = 1;
				VariaveisControle.NUM_OBJECTS =
					Integer.parseInt((String)
					(numObject.getText().toString()));
				VariaveisControle.TIME_TOTAL =
					Long.parseLong((String)
					(tempoTeste.getText().toString())) * 1000;

				Intent myIntent = new Intent(view.getContext(),
						TelaExecucao.class);
				startActivityForResult(myIntent, 0);
			}
		});

		botaoTeste2.setOnClickListener(new View.OnClickListener() {
			public void onClick(View view) {
				VariaveisControle.contFechar++;

				VariaveisControle.TIPO_TESTE = 2;
				VariaveisControle.NUM_OBJECTS =
					Integer.parseInt((String)
					(numObject.getText().toString()));
				VariaveisControle.TIME_TOTAL =
```

```
                        Long.parseLong((String)
                        (tempoTeste.getText().toString())) * 1000;

                Intent myIntent = new Intent(view.getContext(),
                                TelaExecucao.class);
                startActivityForResult(myIntent, 0);
        }
});

botaoTeste3.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
                VariaveisControle.contFechar++;

                VariaveisControle.TIPO_TESTE = 3;
                VariaveisControle.NUM_OBJECTS =
                        Integer.parseInt((String)
                        (numObject.getText().toString()));
                VariaveisControle.TIME_TOTAL =
                        Long.parseLong((String)
                        (tempoTeste.getText().toString())) * 1000;

                Intent myIntent = new Intent(view.getContext(),
                                TelaExecucao.class);
                startActivityForResult(myIntent, 0);
        }
});

botaoTeste4.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
                VariaveisControle.contFechar++;

                VariaveisControle.TIPO_TESTE = 4;
                VariaveisControle.NUM_OBJECTS =
                        Integer.parseInt((String)
                        (numObject.getText().toString()));
                VariaveisControle.TIME_TOTAL =
                        Long.parseLong((String)
                        (tempoTeste.getText().toString())) * 1000;

                Intent myIntent = new Intent(view.getContext(),
                                TelaExecucao.class);
                startActivityForResult(myIntent, 0);
        }
});

botaoTeste5.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
                VariaveisControle.contFechar++;

                VariaveisControle.TIPO_TESTE = 5;
                VariaveisControle.NUM_OBJECTS =
                        Integer.parseInt((String)
                        (numObject.getText().toString()));
                VariaveisControle.TIME_TOTAL =
                        Long.parseLong((String)
                        (tempoTeste.getText().toString())) * 1000;

                Intent myIntent = new Intent(view.getContext(),
                                TelaExecucao.class);
                startActivityForResult(myIntent, 0);
        }
});

botaoTeste6.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
                VariaveisControle.contFechar++;
```

```
                        VariaveisControle.TIPO_TESTE = 6;
                        VariaveisControle.NUM_OBJECTS =
                                Integer.parseInt((String)
                                (numObject.getText().toString()));
                        VariaveisControle.TIME_TOTAL =
                                Long.parseLong((String)
                                (tempoTeste.getText().toString())) * 1000;

                        Intent myIntent = new Intent(view.getContext(),
                                        TelaExecucao.class);
                        startActivityForResult(myIntent, 0);
                }
        });

        botaoSair.setOnClickListener(new View.OnClickListener() {
                public void onClick(View view) {
                        System.exit(0);
                }
        });

}

public void onResume() {
        super.onResume();
        VariaveisControle.FRAMES_TOTAL = 0;
        VariaveisControle.PORC_CPU = 0;

        VariaveisControle.VIEW_COLOR_R = (float) (Math.random());
        VariaveisControle.VIEW_COLOR_G = (float) (Math.random());
        VariaveisControle.VIEW_COLOR_B = (float) (Math.random());
}




public void onPause() {
        super.onPause();

        if (VariaveisControle.contFechar == 0) {
                System.exit(0);
        }
}

}
```

B.2.2 TelaExecucao

Activity onde é exibido todos os objetos após feitas as manipulações.

```
package com.TccNehe;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.Window;
import android.view.WindowManager;

public class TelaExecucao extends Activity {

        private GLSurfaceView surface;
        private OpenGLRender openGLRender;
```

```
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);

                // Cria uma aplicaçao fullscream
                this.requestWindowFeature(Window.FEATURE_NO_TITLE);
                getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                                WindowManager.LayoutParams.FLAG_FULLSCREEN);
                // ----------------------------------

                surface = new GLSurfaceView(this);
                openGLRender = new OpenGLRender(this);
                surface.setRenderer(openGLRender);

                setContentView(surface);
        }

        public void onResume() {
                super.onResume();
                surface.onResume();
        }

        public void onPause() {
                super.onPause();
                finish();
        }

        public boolean onTouchEvent(MotionEvent event) {
                return openGLRender.onTouchEvent(event);
        }
}
```

B.2.3 TelaResultados

Activity usada para a exibição dos resultados obtidos após a execução de cada teste.

```
package com.TccNehe;

import java.text.DecimalFormat;

import android.app.Activity;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;
import android.widget.TextView;

public class TelaResultados extends Activity {

        private TextView tvResultTotalFrames;
        private TextView tvResultTempoTotal;
        private TextView tvResultFPS;
        private TextView tvResultCPU;

        @Override
        public void onCreate(Bundle savedInstanceState) {

                super.onCreate(savedInstanceState);

                // Cria uma aplicaçao fullscream
                this.requestWindowFeature(Window.FEATURE_NO_TITLE);
                getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                                WindowManager.LayoutParams.FLAG_FULLSCREEN);
                // ----------------------------------
```

```
                setContentView(R.layout.resultados);

                DecimalFormat df = new DecimalFormat("0.00");

                tvResultTotalFrames = (TextView) findViewById(R.id.tvResultTotalFrames);
                tvResultTempoTotal = (TextView) findViewById(R.id.tvResultTempoTotal);
                tvResultFPS = (TextView) findViewById(R.id.tvResultFPS);
                tvResultCPU = (TextView) findViewById(R.id.tvResultPorcDeCPU);

                float fps = (float) (VariaveisControle.FRAMES_TOTAL)
                                / (float) (VariaveisControle.TIME_TOTAL / 1000);

                tvResultTotalFrames.setText("" + VariaveisControle.FRAMES_TOTAL);
                tvResultTempoTotal.setText("" + VariaveisControle.TIME_TOTAL / 1000);

                tvResultFPS.setText("" + df.format(fps));
                tvResultCPU.setText("" + df.format(VariaveisControle.PORC_CPU));

        }

        public void onPause() {
                super.onPause();
                VariaveisControle.contFechar--;
        }
}
```

B.3 OpenGLRender

Nesta classe é feito todos os cálculos necessários para a manipulação dos objetos em cena. Também é configurado os parâmetros do OpenGL ES para que seja aplicado os efeitos necessários. Esta classe é ativada a partir da *Activity* TelaExecucao.

```
package com.TccNehe;

import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;
import java.util.Date;
import java.util.Random;
import java.util.Vector;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLU;
import android.opengl.GLUtils;
import android.opengl.GLSurfaceView.Renderer;
import android.util.Log;
import android.view.MotionEvent;

class OpenGLRender implements Renderer {

        private int[] textures = new int[3];
        private int xrot;
```

```
private int yrot;
private int oldX;
private int oldY;

private float posCameraX = 0;
private float posCameraY = 0;
private float posCameraZ = (-VariaveisControle.VIEW_DEPTH / 2) - 2;

private long tempoInicial = 0;
private long tempoFinal = 0;

private CpuUsage porcCPU;

private Vector<Objeto> listaObjetos = new Vector<Objeto>();

public OpenGLRender(Context context) {
	VariaveisControle.CONTEXTO = context;
}

public void onSurfaceCreated(GL10 gl, EGLConfig config) {

	criarObjetos(VariaveisControle.NUM_OBJECTS);

	if (VariaveisControle.TIPO_TESTE == 2) {
		aplicarIluminação(gl);
	}

	if (VariaveisControle.TIPO_TESTE == 3) {
		aplicarFog(gl);
	}

	if (VariaveisControle.TIPO_TESTE == 4) {
		aplicarIluminação(gl);
		aplicarFog(gl);
	}

	if (VariaveisControle.TIPO_TESTE == 5) {
		aplicaTextura(gl);
	}

	if (VariaveisControle.TIPO_TESTE == 6) {
		aplicarIluminação(gl);
		aplicarFog(gl);
		aplicaTextura(gl);
	}

	gl.glEnable(GL10.GL_DEPTH_TEST);
	gl.glEnable(GL10.GL_CULL_FACE);
	gl.glDepthFunc(GL10.GL_LEQUAL);
	gl.glClearColor(VariaveisControle.VIEW_COLOR_R,
			VariaveisControle.VIEW_COLOR_G,
			VariaveisControle.VIEW_COLOR_B,
			VariaveisControle.VIEW_COLOR_A);
	gl.glClearDepthf(1.0f);
	gl.glCullFace(GL10.GL_BACK);
	gl.glShadeModel(GL10.GL_SMOOTH);
	gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
	gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
	gl.glFrontFace(GL10.GL_CW);

	tempoInicial = System.currentTimeMillis();
	porcCPU = new CpuUsage();

}

public void onSurfaceChanged(GL10 gl, int w, int h) {
	gl.glViewport(0, 0, w, h);
```

```
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        GLU.gluPerspective(gl, 90.0f, ((float) w) / h, 1.0f, 50.0f);
        gl.glMatrixMode(GL10.GL_MODELVIEW);

        porcCPU.update();
    }

    public void onDrawFrame(GL10 gl) {

        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(posCameraX, posCameraY, posCameraZ);
        gl.glRotatef(xrot, 1, 0, 0);
        gl.glRotatef(yrot, 0, 1, 0);

        for (int i = 0; i < listaObjetos.size(); i++) {

                listaObjetos.get(i).translate[0] += listaObjetos.get(i).x;
                listaObjetos.get(i).translate[1] += listaObjetos.get(i).y;
                listaObjetos.get(i).translate[2] += listaObjetos.get(i).z;

                listaObjetos.get(i).anguloAtualX += listaObjetos.get(i).anguloX;
                listaObjetos.get(i).anguloAtualY += listaObjetos.get(i).anguloY;
                listaObjetos.get(i).anguloAtualZ += listaObjetos.get(i).anguloZ;

                gl.glFlush();
                gl.glPushMatrix();
                listaObjetos.get(i).drawCube(gl);
                gl.glPopMatrix();

        }

        VariaveisControle.FRAMES_TOTAL++;
        tempoFinal = System.currentTimeMillis();

        if ((tempoFinal - tempoInicial) > VariaveisControle.TIME_TOTAL) {

                VariaveisControle.PORC_CPU = porcCPU.update();
                try {
                        porcCPU.close();
                } catch (IOException e) {
                        Log.w("ERRO", "ERRO AO CALCULAR % DE CPU");
                        e.printStackTrace();
                }

                tempoInicial = tempoFinal;
                VariaveisControle.contFechar++;

                VariaveisControle.CONTEXTO.startActivity(new Intent(
                                VariaveisControle.CONTEXTO,
                                TelaResultados.class));

        }

    }

    public void onPause() {
        VariaveisControle.contFechar--;
    }

    private void criarObjetos(int quantidade) {

        for (int i = 0; i < quantidade; i++) {

                float tamanho = ((float) (Math.random() * 100) + 1) / 150;
```

```
                Log.w("Tamanho", "" + tamanho);

                Objeto objeto = new Objeto(tamanho);
                float x = VariaveisControle.VIEW_WIDTH;
                float y = VariaveisControle.VIEW_HEIGHT;
                float z = VariaveisControle.VIEW_DEPTH;

                objeto.cor[0] = (float) Math.random();
                objeto.cor[1] = (float) Math.random();
                objeto.cor[2] = (float) Math.random();

                objeto.translate[0] = ((float) Math.random() * x) - (x / 2);
                objeto.translate[1] = ((float) Math.random() * y) - (y / 2);
                objeto.translate[2] = ((float) Math.random() * z) - (z / 2);

                objeto.x = ((float) Math.random()) / 2;
                objeto.y = ((float) Math.random()) / 2;
                objeto.z = ((float) Math.random()) / 2;

                objeto.anguloX = ((float) Math.random() * 3) - 3;
                objeto.anguloY = ((float) Math.random() * 3) - 3;
                objeto.anguloZ = ((float) Math.random() * 3) - 3;

                listaObjetos.add(objeto);
        }

}

public boolean onTouchEvent(MotionEvent event) {
        int x = (int) event.getX();
        int y = (int) event.getY();
        if (event.getAction() == MotionEvent.ACTION_MOVE) {
                xrot -= (oldY - y);
                yrot += (x - oldX);
        }
        oldX = x;
        oldY = y;
        return true;
}

public ByteBuffer getImageBuffer(int resID) {
        Bitmap bmp = BitmapFactory.decodeResource(
                        VariaveisControle.CONTEXTO.getResources(), resID);
        ByteBuffer bb = ByteBuffer.allocateDirect(bmp.getHeight()
                        * bmp.getWidth() * 4);
        bb.order(ByteOrder.BIG_ENDIAN);
        IntBuffer ib = bb.asIntBuffer();

        for (int y = 0; y < bmp.getHeight(); y++)
                for (int x = 0; x < bmp.getWidth(); x++) {
                        int pix = bmp.getPixel(x, bmp.getHeight() - y - 1);
                        byte alpha = (byte) ((pix >> 24) & 0xFF);
                        byte red = (byte) ((pix >> 16) & 0xFF);
                        byte green = (byte) ((pix >> 8) & 0xFF);
                        byte blue = (byte) ((pix) & 0xFF);

                        ib.put(((red & 0xFF) << 24) | ((green & 0xFF) << 16)
                                        | ((blue & 0xFF) << 8) | ((alpha &
                                        0xFF)));
                }
        ib.position(0);
        bb.position(0);
        bmp.recycle();
        return bb;
}

private void aplicarIluminação(GL10 gl) {
```

```
Date date = new Date();
long time = date.getTime();
Random num = new Random(time);

// ************************************************************
// priemiro ponto de luz

float matEspecular1[] = { num.nextFloat(), num.nextFloat(),
              num.nextFloat(), 1 };
float lAmbiente1[] = { num.nextFloat(), num.nextFloat(),
              num.nextFloat(), 1 };
float lDifusa1[] = { num.nextFloat(), num.nextFloat(), num.nextFloat(),
              1 };
float lEspecular1[] = { num.nextFloat(), num.nextFloat(),
              num.nextFloat(), 1 };
float shininess1 = num.nextFloat() * 80;
float posicaoLuz1[] = {
              (num.nextFloat() * (VariaveisControle.VIEW_WIDTH -
              VariaveisControle.VIEW_EDGE))-
              ((VariaveisControle.VIEW_WIDTH -
              VariaveisControle.VIEW_EDGE) / 2),
              (num.nextFloat() * (VariaveisControle.VIEW_HEIGHT -
              VariaveisControle.VIEW_EDGE)) -
              ((VariaveisControle.VIEW_HEIGHT -
              VariaveisControle.VIEW_EDGE) / 2),
              (num.nextFloat() * VariaveisControle.VIEW_DEPTH), 1 };

gl.glShadeModel(GL10.GL_SMOOTH);
gl.glEnable(GL10.GL_COLOR_MATERIAL);

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, lAmbiente1, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, lDifusa1, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
              matEspecular1, 0);
gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininess1);

gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, lAmbiente1, 0);
gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, lDifusa1, 0);
gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_SPECULAR, lEspecular1, 0);
gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, posicaoLuz1, 0);

// ************************************************************
// segundo ponto de luz

float matEspecular2[] = { num.nextFloat(), num.nextFloat(),
              num.nextFloat(), 1 };
float lAmbiente2[] = { num.nextFloat(), num.nextFloat(),
              num.nextFloat(), 1 };
float lDifusa2[] = { num.nextFloat(), num.nextFloat(), num.nextFloat(),
              1 };
float lEspecular2[] = { num.nextFloat(), num.nextFloat(),
              num.nextFloat(), 1 };
float shininess2 = num.nextFloat() * 80;
float posicaoLuz2[] = {
              (num.nextFloat() * (VariaveisControle.VIEW_WIDTH -
              VariaveisControle.VIEW_EDGE)) -
              ((VariaveisControle.VIEW_WIDTH -
              VariaveisControle.VIEW_EDGE) / 2),
              (num.nextFloat() * (VariaveisControle.VIEW_HEIGHT -
              VariaveisControle.VIEW_EDGE)) -
              ((VariaveisControle.VIEW_HEIGHT -
              VariaveisControle.VIEW_EDGE) / 2),
              (num.nextFloat() * VariaveisControle.VIEW_DEPTH), 1 };

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, lAmbiente2, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, lDifusa2, 0);
```

```
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
                matEspecular2, 0);
gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininess2);

gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lAmbiente2, 0);
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lDifusa2, 0);
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_SPECULAR, lEspecular2, 0);
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, posicaoLuz2, 0);

// *************************************************************
// terceiro ponto de luz

float matEspecular3[] = { num.nextFloat(), num.nextFloat(),
                num.nextFloat(), 1 };
float lAmbiente3[] = { num.nextFloat(), num.nextFloat(),
                num.nextFloat(), 1 };
float lDifusa3[] = { num.nextFloat(), num.nextFloat(), num.nextFloat(),
                1 };
float lEspecular3[] = { num.nextFloat(), num.nextFloat(),
                num.nextFloat(), 1 };
float shininess3 = num.nextFloat() * 80;
float posicaoLuz3[] = {
                (num.nextFloat() * (VariaveisControle.VIEW_WIDTH -
                VariaveisControle.VIEW_EDGE)) -
                ((VariaveisControle.VIEW_WIDTH -
                VariaveisControle.VIEW_EDGE) / 2),
                (num.nextFloat() * (VariaveisControle.VIEW_HEIGHT -
                VariaveisControle.VIEW_EDGE)) -
                ((VariaveisControle.VIEW_HEIGHT -
                VariaveisControle.VIEW_EDGE) / 2),
                (num.nextFloat() * VariaveisControle.VIEW_DEPTH), 1 };

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, lAmbiente3, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, lDifusa3, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
                matEspecular3, 0);
gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininess3);

gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lAmbiente3, 0);
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lDifusa3, 0);
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_SPECULAR, lEspecular3, 0);
gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, posicaoLuz3, 0);

// *************************************************************
// quarto ponto de luz

float matEspecular4[] = { num.nextFloat(), num.nextFloat(),
                num.nextFloat(), 1 };
float lAmbiente4[] = { num.nextFloat(), num.nextFloat(),
                num.nextFloat(), 1 };
float lDifusa4[] = { num.nextFloat(), num.nextFloat(), num.nextFloat(),
                1 };
float lEspecular4[] = { num.nextFloat(), num.nextFloat(),
                num.nextFloat(), 1 };
float shininess4 = num.nextFloat() * 80;
float posicaoLuz4[] = {
                (num.nextFloat() * (VariaveisControle.VIEW_WIDTH -
                VariaveisControle.VIEW_EDGE)) -
                ((VariaveisControle.VIEW_WIDTH -
                VariaveisControle.VIEW_EDGE) / 2),
                (num.nextFloat() * (VariaveisControle.VIEW_HEIGHT -
                VariaveisControle.VIEW_EDGE)) -
                ((VariaveisControle.VIEW_HEIGHT -
                VariaveisControle.VIEW_EDGE) / 2),
                (num.nextFloat() * VariaveisControle.VIEW_DEPTH), 1 };

gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, lAmbiente4, 0);
```

```
        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, lDifusa4, 0);
        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
                        matEspecular4, 0);
        gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininess4);

        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lAmbiente4, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lDifusa4, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_SPECULAR, lEspecular4, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, posicaoLuz4, 0);

        // ************************************************************
        // quinto ponto de luz

        float matEspecular5[] = { num.nextFloat(), num.nextFloat(),
                        num.nextFloat(), 1 };
        float lAmbiente5[] = { num.nextFloat(), num.nextFloat(),
                        num.nextFloat(), 1 };
        float lDifusa5[] = { num.nextFloat(), num.nextFloat(), num.nextFloat(),
                        1 };
        float lEspecular5[] = { num.nextFloat(), num.nextFloat(),
                        num.nextFloat(), 1 };
        float shininess5 = num.nextFloat() * 80;
        float posicaoLuz5[] = {
                        (num.nextFloat() * (VariaveisControle.VIEW_WIDTH -
                        VariaveisControle.VIEW_EDGE))-
                        ((VariaveisControle.VIEW_WIDTH -
                        VariaveisControle.VIEW_EDGE) / 2),
                        (num.nextFloat() * (VariaveisControle.VIEW_HEIGHT -
                        VariaveisControle.VIEW_EDGE))-
                        ((VariaveisControle.VIEW_HEIGHT -
                        VariaveisControle.VIEW_EDGE) / 2),
                        (num.nextFloat() * VariaveisControle.VIEW_DEPTH), 1 };

        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, lAmbiente5, 0);
        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, lDifusa5, 0);
        gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_SPECULAR,
                        matEspecular5, 0);
        gl.glMaterialf(GL10.GL_FRONT_AND_BACK, GL10.GL_SHININESS, shininess5);

        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_AMBIENT, lAmbiente5, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_DIFFUSE, lDifusa5, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_SPECULAR, lEspecular5, 0);
        gl.glLightfv(GL10.GL_LIGHT1, GL10.GL_POSITION, posicaoLuz5, 0);

        // ************************************************************

        gl.glEnable(GL10.GL_LIGHTING);
        gl.glEnable(GL10.GL_LIGHT0);
        gl.glEnable(GL10.GL_LIGHT1);
        gl.glEnable(GL10.GL_LIGHT2);
        gl.glEnable(GL10.GL_LIGHT3);
        gl.glEnable(GL10.GL_LIGHT4);
        gl.glEnable(GL10.GL_DEPTH_TEST);

    }

    private void aplicaTextura(GL10 gl) {
        gl.glEnable(GL10.GL_TEXTURE_2D);
        loadTexture(gl);
    }

    private void aplicarFog(GL10 gl) {
        float[] fogColor = { VariaveisControle.VIEW_COLOR_R,
                        VariaveisControle.VIEW_COLOR_G,
                        VariaveisControle.VIEW_COLOR_B,
                        VariaveisControle.VIEW_COLOR_A };
```

```
                gl.glFogfv(GL10.GL_FOG_COLOR, fogColor, 0);
                gl.glFogf(GL10.GL_FOG_DENSITY, 0.8f);
                gl.glFogx(GL10.GL_FOG_MODE, GL10.GL_LINEAR);

                gl.glFogf(GL10.GL_FOG_START, VariaveisControle.START_FOG);
                gl.glFogf(GL10.GL_FOG_END, VariaveisControle.FINISH_FOG);

                gl.glHint(GL10.GL_FOG_HINT, GL10.GL_NICEST);
                gl.glEnable(GL10.GL_FOG);

        }

        protected void loadTexture(GL10 gl) {
                gl.glGenTextures(2, textures, 0);

                textures[0] = R.drawable.imagem1;
                textures[1] = R.drawable.imagem2;
                textures[2] = R.drawable.imagem3;

                gl.glActiveTexture(GL10.GL_TEXTURE0);
                gl.glClientActiveTexture(GL10.GL_TEXTURE0);
                gl.glEnable(GL10.GL_TEXTURE_2D);
                gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[0]);
                gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[1]);
                gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[2]);
                gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
                gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, listaObjetos.get(0).text);
                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
                                GL10.GL_LINEAR);
                gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
                                GL10.GL_LINEAR);
                gl.glTexEnvf(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
                                GL10.GL_MODULATE);
                InputStream is = VariaveisControle.CONTEXTO.getResources()
                                .openRawResource(textures[(int) (Math.random() * 3)]);
                Bitmap bitmap = null;
                try {
                        bitmap = BitmapFactory.decodeStream(is);
                } finally {
                        try {
                                is.close();
                        } catch (Exception e) {
                        }
                }
                GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
                bitmap.recycle();

        }

}
```

B.4 Objeto

A classe Objeto possui apenas as coordenadas dos vértices de cada objeto (cubo), as coordenadas da textura a ser aplicada e cada um deles, assim como os índices do objeto, que indicam a sequência de pontos que deve ser desenhadas. Nesta classe também ocorre a renderização de cada objeto na cena, aplicando os efeitos de rotação e translação.

```
package com.TccNehe;
```

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;

import javax.microedition.khronos.opengles.GL10;

public class Objeto {

	public IntBuffer vert;
	public IntBuffer norm;
	public IntBuffer text;
	public ByteBuffer ind;

	public float x = 0;
	public float y = 0;
	public float z = 0;

	public float cor[] = {1,0,1,0};
	public float translate[] = {0, 0, 0};

	public float rotacionar[] = {0, 0, 0};

	public float anguloX = 0;
	public float anguloY = 0;
	public float anguloZ = 0;

	public float anguloAtualX = 0;
	public float anguloAtualY = 0;
	public float anguloAtualZ = 0;

	public Objeto (float lado) {

		int I=toInt(lado);

		int vertices[] = {
				-I,-I,-I,I,-I,-I,I,I,-I,-I,I,-I,
				-I,-I,I,I,-I,I,I,I,-I,I,I,

				-I,-I,-I,I,-I,-I,I,I,-I,-I,I,-I,
				-I,-I,I,I,-I,I,I,I,-I,I,I,

				-I,-I,-I, I,-I,-I, I,I,-I, -I,I,-I,
				-I,-I,I, I,-I,I, I,I,I, -I,I,I
		};
//		int normals[] = {
//				-I,0,0, I,0,0, I,0,0, -I,0,0,
//				-I,0,0, I,0,0, I,0,0, -I,0,0,
//
//				0,-I,0, 0,-I,0, 0,I,0, 0,I,0,
//				0,-I,0, 0,-I,0, 0,I,0, 0,I,0,
//
//				0,0,-I, 0,0,-I, 0,0,I, 0,0,I,
//				0,0,-I, 0,0,-I, 0,0,I, 0,0,I
//		};
		int texture[] = {
				0,0, toInt(1),0, toInt(1),toInt(1), 0,toInt(1),
				toInt(1),0, 0,0, 0,toInt(1), toInt(1),toInt(1),

				0,0, toInt(1),0, toInt(1),toInt(1), 0,toInt(1),
				0,toInt(1), toInt(1),toInt(1), toInt(1),0, 0,0,

				0,0, toInt(1),0, toInt(1),toInt(1), 0,toInt(1),
				toInt(1),0, 0,0, 0,toInt(1), toInt(1),toInt(1)
		};
		byte indices[] = {
				1,5,6, 1,6,2,
				0,3,7, 0,7,4,
```

```
                                13,9,8, 13,8,12,
                                11,10,14, 11,14,15,

                                17,18,19, 17,19,16,
                                21,20,23, 21,23,22
                        };

                        vert=getIntBuffer(vertices);
//                      norm=getIntBuffer(normals);
                        text=getIntBuffer(texture);
                        ind=getByteBuffer(indices);
                }

                public void drawCube(GL10 gl){

                        if (translate[0] >= 13 || translate[0] < -13 ) {
                                x = -x;
                        }

                        if (translate[1] >= 18 || translate[1] < -18 ) {
                                y = -y;
                        }

                        if (translate[2] >= 8 || translate[2] < -8 ) {
                                z = -z;
                        }

                        gl.glTranslatef(translate[0], translate[1], translate[2]);

                        gl.glRotatef(anguloAtualX, 1, 0, 0);
                        gl.glRotatef(anguloAtualY, 0, 1, 0);
                        gl.glRotatef(anguloAtualZ, 0, 0, 1);

                 gl.glColor4f(cor[0], cor[1], cor[2], cor[3]); // Aplica cor ao objeto
                 gl.glVertexPointer(3, GL10.GL_FIXED, 0, vert);
                 gl.glNormalPointer(GL10.GL_FIXED, 0, norm);
                 gl.glDrawElements(GL10.GL_TRIANGLES, 36, GL10.GL_UNSIGNED_BYTE, ind);

        }

        private IntBuffer getIntBuffer(int[] data){
                ByteBuffer byteBuf = ByteBuffer.allocateDirect(data.length * 4);
                byteBuf.order(ByteOrder.nativeOrder());
                IntBuffer buf = byteBuf.asIntBuffer();
                buf.put(data);
                buf.position(0);
                return buf;
        }
        private ByteBuffer getByteBuffer(byte[] data){
                ByteBuffer buf = ByteBuffer.allocateDirect(data.length);
                        buf.put(data);
                        buf.position(0);
                        return buf;
        }

                private int toInt(float num){
                return (int)(num*65536);
        }
}
```

B.5 CpuUsage

Classe onde é calculada a porcentagem de CPU utilizada pela a plicação.

```
package com.TccNehe;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;

import android.util.Log;

public class CpuUsage {
	private static final String TAG = "CpuUsage";
	private RandomAccessFile statFile;
	private long lastTotal;
	private long lastIdle;
	private float usage;

	public CpuUsage() {
		try {
			statFile = new RandomAccessFile("/proc/stat", "r");
		} catch (FileNotFoundException e) {
			statFile = null;
			Log.e(TAG, "oh, cannot open /proc/stat: " + e);
		}
	}

	public void close() throws IOException {
		statFile.close();
	}

	public float update() {
		if (statFile == null)
			return usage;

		try {
			statFile.seek(0);
			String cpuLine = statFile.readLine();
			String[] parts = cpuLine.split("[ ]+");

			if (!"cpu".equals(parts[0])) {
				throw new IllegalArgumentException("unable to get cpu
				line");
			}

			long idle = Long.parseLong(parts[4], 10);
			long total = 0;

			boolean head = true;
			for (String part : parts) {
				if (head) {
					head = false;
					continue;
				}
				total += Long.parseLong(part, 10);
			}

			long diffIdle = idle - lastIdle;
			long diffTotal = total - lastTotal;

			usage = (float) (diffTotal - diffIdle) / diffTotal * 100;
			if (Float.isNaN(usage)) {
				usage = 0;
			}

			lastTotal = total;
			lastIdle = idle;

		} catch (IOException e) {
			Log.e(TAG, "Erro: " + e);
```

```
                }

                return usage;
        }

}
```

B.5 VariaveisControle

Classe utilizada para armazenar algumas variáveis utilizadas na aplicação.

```
package com.TccNehe;

import android.content.Context;

public class VariaveisControle {

        // Dimensao da Janela
        public static final float VIEW_WIDTH = 26;
        public static final float VIEW_HEIGHT = 36;
        public static final float VIEW_DEPTH = 16;
        public static final float VIEW_EDGE = 3;

        public static float VIEW_COLOR_R = (float) (Math.random());
        public static float VIEW_COLOR_G = (float) (Math.random());
        public static float VIEW_COLOR_B = (float) (Math.random());
        public static float VIEW_COLOR_A = 0;

        // Variáveis para testes
        public static long TIME_TOTAL = 120000;
        public static long FRAMES_TOTAL = 0;
        public static float PORC_CPU = 0;

        // variaveis de controle
        public static Context CONTEXTO;
        public static int TIPO_TESTE = 6;
        public static int NUM_OBJECTS = 50;

        public static int contFechar = 0;

        public static final float START_FOG = (-VIEW_DEPTH / 2) - 2;
        public static final float FINISH_FOG = (VIEW_DEPTH / 1.5f)
                        + (float) (Math.random() * (VIEW_DEPTH / 3));

}
```

Referências Bibliográficas

[1] Dumitru, B, C. History of Android. Disponível em: <http://bogdancdumitru.blogspot.com/p/history-of-android.html>. Acesso em: 06 de Julho de 2012

[2] Yadav, M. History of Android. Dezembro de 2011. Disponível em: <http://www.tech2crack.com/history-android/>. Acesso em: 05 de Julho de 2012

[3] Zechner, M. Beginning Android Games, 1. ed. USA: Apress, 2011

[4] Yackulic, C. The History of Android, A True Story of Success. Setembro 2010. Disponível em: <http://androidheadlines.com/2010/09/the-history-of-android-a-true-story-of-success.htm>. Acesso em: 02 de Julho de 2012

[5] The Android Story. Disponível em: <http://www.xcubelabs.com/the-android-story.php>. Acesso em: 04 de Julho de 2012

[6] comScore. Reports March 2012 U.S. Mobile Subscriber Market Share. Maio de 2012. Disponível em: <http://www.comscore.com/Press_Events/Press_Releases/2012/5/comScore_Reports _March_2012_U.S._Mobile_Subscriber_Market_Share>. Acesso em: 10 de Julho de 2012

[7] Higa, P. Android ultrapassa 50% de market share; Samsung vende mais. Abril de 2012. Disponível em: <http://tecnoblog.net/97184/android-market-share/>. Acesso em: 08 de Julho de 2012

[8] Darcey, L, Conder, S. Android Application Development in 24 Hours. 1. ed. Indiana: Sams 2010

[9] Open Handset Alliance. Open Handset Alliance Members. Disponível em: <http:// www.openhandsetalliance.com/oha_members.html>. Acesso em: 07 de Julho de 2012

[10] Luis, A. Vantagens do android sobre iPhone. Julho de 2011. Disponível em: <http://www.celularesandroid.com.br/dicas-e-tutoriais/vantagens-do-android-sobre-o-iphone/>. Acesso em: 03 de Julho de 2012

[11] Ultra Downloads. Conhece todas as versões de Android?. Janeiro de 2012. Disponível em: < http://ultradownloads.com.br/dica/Conhece-todas-as-versoes-de-Android/>. Acesso em: 05 de Julho de 2012

[12] Wikipedia, Android version history. Outubro de 2012. Deisponível em <http://en.wikipedia.org/wiki/Jelly_Bean_(operating_system)#Android_4.1.x_Jelly_Bean>. Acesso em: 16 de Outubro de 2012

[13] DiMarzio, J. Android: A Programmer's Guide. 1. ed. EUA: McGraw-Hill, 2008

[14] Rogers, R, Lombardo, J, Mednieks, Z, Meike, B. Android Application Development. 1. ed. USA: O'REILLY, 2009

[15] Open Handset Alliance. Open Handset Alliance. Disponível em: <http://www.openhandsetalliance.com/>. Acesso em: 05 de Julho de 2012

[16] Lee, W. Beginning Android Application Development. 1. ed. Indiana: Wiley, 2011

[17] Tudocelular.com. Abril de 2011. Disponível em: <Froyo, http://www.tudocelular.com/Curiosidade/noticias/n24075/froyo-android-66.html>. Acesso em: 06 de Julho de 2012

[18] Android Developers. Março de 2012. Android APIs levels. Disponível em: <http://developer.android.com/guide/appendix/api-levels.html>. Acesso em: 11 de Julho 2012

[19] Khronos. OpenGL ES. Disponível em: <http://www.khronos.org/opengles/>. Acesso em: 10 de Julho de 2012

[20] Wenderlich, R. Beginning OpenGL ES 2.0. Outubro de 2011. Disponível em:<http://www.raywenderlich.com/5223/beginning-opengl-es-2-0-with-glkit-part-1>. Acesso em: 9 de Julho de 2012

[21] Android Open Source Project. Disponível em: <http://source.android.com/compatibility/overview.html>. Acesso em: 10 de Agosto de 2012

[22] Wikipedia, OpenGL. Disponível em: <http://pt.wikipedia.org/wiki/OpenGL>. Acesso em: 20 de Stembro de 2012

[23] Segal, M, Akeley, K. The OpenGLR Graphics System: A Specification (Version 4). Março de 2010 (http://www.opengl.org/registry/doc/glspec40.core.20100311.pdf)

[24] Wikipedia, OpenGL. Disponível em: <http://en.wikipedia.org/wiki/OpenGL>. Acesso em: 16 de Outubro de 2012

[25]Who, M, Neider, J, Davis, T. Opengl Programming Guide: The Official Guide to Learning Opengl, Version 1.1. Disponível em: <http://www.glprogramming.com/red/chapter01.html>. Acesso em: 5 de Setembro de 2012

[26] Kishimoto, A. GPU e Shaders. Disponível em: <http://www.tupinihon.com/tech/pdf/artigo-gpu.pdf >. Acesso em 8 de Outubro de 2012.

[27] Wikipedia. OpenGL ES. Junho 2012. disponível em: <http://en.wikipedia.org/wiki/OpenGL_ES>. Acesso em: 11 de Julho de 2012

[28] Armasu, L. OpenGL ES 3.0 “Haiti” to be released this summer. Junho de 2012. Disponível em: <http://www.androidauthority.com/opengl-es-3-0-haiti-to-be-released-this-summer-98192/>, Acesso em: 10 de Julho de 2012

[29] Robert. Rendering Pipeline. Março de 2012. Disponível em: <http://renderingpipeline .com/2012/03/opengl-es-3-0/>. Acesso em: 12 de Julho de 2012

[30] Miranda, G. O que é Pixel Shader e Vertex Shader. Fevereiro 2012. Disponível em: <http://www.blogplus.com.br/?p=12413>. Acesso em: 10 de Julho de 2012

[31] Wikipedia. Fevereiro de 2012. Disponível em: <http://pt.wikipedia.org/wiki /Vertex_shader>. Acesso em: 13 de Julho de 2012

[32] Wikipedia. Shader. Junho de 2012. Disponível em: <http://en.wikipedia.org/ wiki/Shader>. Acesso em: 13 de Julho de 2012

[33] Wikipedia. Shader (realtime, logical). Abril de 2012. Disponível em: <http://en.wikipedia.org/wiki/Shader_(realtime,_logical)>. Acesso em: 10 de Julho de 2012

[34] Vasconcelos, C, N. Algoritmos para Processamento de Imagens e Visão Computacional para Arquiteturas Paralelas em Placas Gráficas. Tese de Doutorado. Rio de Janeiro, 2009

[35] Guilherme, P. O que é pixel shader?. Maio de 2012. Disponível em: <http://www.tecmundo.com.br/placa-de-video/811-o-que-e-pixel-shader-.htm>. Acesso em: 11 de Julho de 2012

[36] nvidia. Pixel Shader. Disponível em: <http://www.nvidia.com/object/feature_ pixelshader.html>. Acesso em: 12 de Julho de 2012

[37] techtudo. Custom roms no android sai o android de fabrica entra outro com sua cara. Novembro de 2011. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2011/11/custom-roms-no-android-sai-o-android-de-fabrica-entra-outro-com-sua-cara.html>

[38] Calandrini, A. O que são Custom Roms? [Android]. Fevereiro de 2011. Disponível em: <http://vidamovelblog.com/2011/02/o-que-sao-custom-roms-android.html>. Acesso em: 3 de Outubro de 2012.