Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

3DViewer: Visualizador de Imagens Tridimensionais para o Sistema Operacional Android

Autor:

____
Guilherme Barbosa Filgueiras

Orientador:

____
Prof. Sérgio Barbosa Villas Boas, D. Sc.

Examinador:

____
Prof. Jorge Lopes de Souza Leão, D. Sc.

Examinador:

____
Prof. Edilberto Strauss, D. Sc.

DEL

Agosto de 2009

DEDICATÓRIA

Dedico este trabalho aos meus pais e à minha noiva Raquel pelo apoio que sempre me deram e por compreenderem a minha ausência até a conclusão de mais essa etapa em minha vida.

AGRADECIMENTO

Agradeço ao povo brasileiro que contribuiu de forma significativa à minha formação e estada nesta Universidade. Este projeto é uma pequena forma de retribuir o investimento e confiança em mim depositados.

RESUMO

Este projeto tem como objetivo o desenvolvimento de um software renderizador (visualizador) de objetos 3D para o sistema operacional Android.

O Android é um sistema operacional para dispositivos móveis desenvolvido pela empresa Google e lançado em novembro de 2007. Os aplicativos devem ser feitos utilizando a linguagem de programação Java e o plugin integrado ao ambiente de desenvolvimento Eclipse.

Dentre os vários sistemas operacionais existentes hoje, o Android foi escolhido por ser baseado em um *kernel* Linux e seguir a filosofia *open source*.

Palavras-Chave: 3DViewer, OpenGL, Android, Google, Visualizador.

ABSTRACT

This project aims at developing a renderer software (viewer) of 3D objects for the Android operating system.

The Android is an operating system for mobile devices developed and launched by Google in November 2007. The applications should be made using the Java programming language together with the integrated Android Eclipse Plugin (Android Development Tools).

Among the various operating systems available today, the Android was chosen because is based on a Linux kernel and follows the open source philosophy.

Key-words: 3DViewer, OpenGL, Android, Google, Viewer.

SIGLAS

UFRJ – Universidade Federal do Rio de Janeiro

WYSIWYG – *What you see is what you get*

DXF - *Drawing Interchange Format* ou *Drawing Exchange Format*

ACI – *AutoCAD Color Index*

API - *Application Programming Interface*

OpenGL – *Open Graphics Library*

OpenGL ES – *Open Graphics Library for Embedded Systems*

SDK – *Software Development Kit*

ADT – Android Development Tools

UI – *User Interface*

IDE - *Integrated Development Environment*

JNI – *Java Native Interface*

RGBA – *Red Green Blue Alpha*

Sumário

1 Introdução 1

1.1 - Dispositivos Móveis e os Computadores....................1
1.2 - Cenário Atual....................1
1.3 - Por que usar o Android?....................2

2 Tecnologias Utilizadas 3

2.1 - Sistema Operacional Android....................3
2.1.1 - O que é o Android?....................3
2.1.2 - Arquitetura do Android....................4
2.1.2.1 - Aplicativos....................5
2.1.2.2 - Framework dos Aplicativos....................5
2.1.2.3 - Bibliotecas....................6
2.1.2.4 - Android Runtime....................7
2.1.2.5 - Kernel Linux....................7
2.1.3 - Ambiente de Desenvolvimento....................7
2.1.3.1 - Requisitos Mínimos....................7
2.1.3.2 - Instalação....................8
2.1.3.3 - Android Development Tools Plugin....................9
2.1.4 - Componentes de um Aplicativo no Android....................11
2.1.5 - Construindo o Primeiro Aplicativo: “Hello, World”....................12
2.1.5.1 - Criando um Novo Projeto....................13
2.1.5.2 - Construindo a Interface do Usuário....................14
2.1.5.3 - Executando o Aplicativo....................14
2.1.5.4 - Utilizando um Layout XML na UI....................15
2.1.5.5 - Utilizando o Debugger no Projeto....................15
2.2 - OpenGL....................15
2.2.1 - Introdução....................15
2.2.2 - OpenGL ES....................16
2.2.3 - Funcionamento Básico....................16
2.3 - Arquivos DXF....................17

3 O 3DViewer 21

3.1 - Introdução....................21
3.2 - A Solução Escolhida....................22
3.3 - Desenvolvimento Inicial....................25
3.4 - Criação de Funções....................37
3.5 - Resultados....................39

4 Conclusão 42

Bibliografia 44

Lista de Figuras

Figura 2.1 - Componentes do sistema operacional Android..3
Figura 2.2 - Ciclo de vida de uma Activity..10
Figura 2.3 - Exemplo de parte de um arquivo DXF..17
Figura 3.1 - Software de exemplo da biblioteca wxWidgets.................................20
Figura 3.2 - Acesso ao linux emulado do Android..21
Figura 3.3 - Importar ou exportar um arquivo para o emulador do Android através do Eclipse..23
Figura 3.4 - Estrutura de um projeto do Android no Eclipse.................................24
Figura 3.5 - Configuração da projeção perspectiva...34
Figura 3.6 - Tela do 3DViewer...34
Figura 3.7 - Menu do 3DViewer para a escolha do arquivo DXF.........................35
Figura 3.8 - Tela inicial do 3DViewer com as instruções de uso...........................38
Figura 3.9 - Barra de progresso do 3DViewer...38
Figura 3.10 - Tela principal do 3DViewer com um arquivo DXF carregado.........39

Lista de Tabelas

Tabela 2.1 - Instalando o Android Development Tools no Eclipse...................................8

Capítulo 1
Introdução

1.1 - Dispositivos Móveis e os Computadores

Os dispositivos móveis estão cada vez mais populares e agregando funções que vão muito além de uma simples agenda eletrônica ou de um meio de comunicação, como os celulares. Esses dispositivos estão se tornando pequenos computadores que realizam tarefas complexas com um poder de processamento que cresce continuamente.

O comércio eletrônico através desses dispositivos é comumente chamado de *mobile commerce* ou simplesmente *m-commerce* em alusão ao já tão conhecido *e-commerce*. Esse tipo de comércio existe em várias modalidades como tíquetes, cupons, ingressos e pontos de fidelidade. O atrativo é justamente colocar tudo isso dentro de um celular, por exemplo.

Assim, empresas de todos os setores buscam aumentar o número de clientes através de propagandas e facilidades dentro de dispositivos móveis. Por tudo isso, esse mercado é atraente também para as empresas de tecnologia.

1.2 - Cenário Atual

Hoje há dezenas de modelos diferentes de celular no mercado e cada um deles pode ter um destes sistemas operacionais: Windows Mobile, iPhone OS, Blackberry, Symbian, Palm OS, alguma distribuição Linux ou o Android. Este último lançado em novembro de 2007 pela Google.

Porém, o desenvolvimento de software nessa área sempre foi considerado arriscado, pois não há um sistema operacional que realmente domine uma grande fatia do mercado como existe nos computadores pessoais. Além disso, a falta de padronização impede a criação de algo que contemple qualquer sistema.

Portanto, qual sistema operacional um desenvolvedor deveria escolher para lançar o seu software? Como saber se um sistema vai ou não ter sucesso no mercado? As respostas certamente não são fáceis, mas é possível tentar seguir uma direção com base no que cada sistema oferece.

1.3 - Por que usar o Android?

Alguns motivos podem ser enumerados. A empresa responsável pelo desenvolvimento do Android é a Google. Simplesmente uma das maiores empresas de tecnologia do mundo e que se destaca por lançar produtos e serviços que fazem sucesso.

Além disso, o Android é gratuito, de código aberto e baseado em um *kernel* Linux.

Por fim, juntamente com o Android, é disponibilizado um SDK para a construção de aplicativos em Java e um plugin para o Eclipse IDE. Java é uma linguagem de programação sólida e bem aceita no mercado e o Eclipse é a IDE mais utilizada no mundo para o desenvolvimento nessa linguagem.

Essa foi a motivação para a criação do 3DViewer: um software de visualização de objetos tridimensionais. A renderização é feita a partir de um arquivo texto gerado pelo *software* AutoCAD utilizando a biblioteca OpenGL. Após ser mostrado na tela, o objeto pode ser rotacionado através dos botões do dispositivo ou ainda passando o dedo sobre a tela, caso a mesma seja sensível ao toque.

No próximo capítulo, é apresentada a arquitetura do Android, os passos para se começar um novo projeto, incluindo as ferramentas necessárias, como conseguí-las e configurá-las, a anatomia de uma aplicação do Android, bem como alguns exemplos de utilização da API disponível no SDK.

Ainda nesse capítulo, o leitor encontrará explicações básicas sobre o que é OpenGL e arquivos DXF. Pois foram tecnologias utilizadas no desenvolvimento do 3DViewer.

No capítulo seguinte, está descrito o processo de desenvolvimento do 3DViewer, quais são e como foram implementadas suas funcionalidades e suas limitações.

Por fim, o autor comenta sobre sua experiência com o Android, seus prós e contras e o que poderia ser feito no futuro para melhorar o 3DViewer.

Capítulo 2

Tecnologias Utilizadas

2.1 - Sistema Operacional Android

Nas seções subsequentes, são apresentadas algumas das particularidades do sistema operacional Android que foram importantes para o desenvolvimento do 3DViewer. Além disso, é explicado como configurar o ambiente de desenvolvimento e os passos a serem seguidos para que se possa começar a construir um aplicativo que rode nesse sistema.

2.1.1 - O que é o *Android*?

O *Android*, na verdade, não se trata apenas de um sistema operacional para dispositivos móveis. Ele também inclui um *middleware* (programa de computador que faz a mediação entre outros softwares) e algumas aplicações básicas que garantem o seu funcionamento.

Abaixo estão listados os recursos oferecidos pelo Android:

- *Framework* de aplicação que possibilita a reutilização e substituição dos componentes.
- Máquina virtual otimizada para dispositivos móveis.
- Navegador integrado baseado no *WebKit*, cujo código fonte é aberto.
- Biblioteca gráfica 2D. Gráficos 3D baseados na especificação OpenGL ES 1.0 (com aceleração de *hardware* opcional).
- SQLite para armazenamento estruturado de dados.
- Suporte para os formatos de áudio, vídeo e imagem mais utilizados (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- Telefonia GSM (dependente do *hardware*).
- Bluetooth, EDGE, 3G e WiFi (dependente do *hardware*).
- Câmera, GPS, bússola e acelerômetro (dependente do *hardware*)

- Ambiente de desenvolvimento rico incluindo um emulador de dispositivo, ferramentas para *debug*, memória e performance e um *plugin* para o Eclipse IDE.

2.1.2 - Arquitetura do Android

O sistema operacional possui alguns componentes principais conforme a figura 2.1 mostra. Em seguida, é possível ver uma descrição detalhada de cada seção.

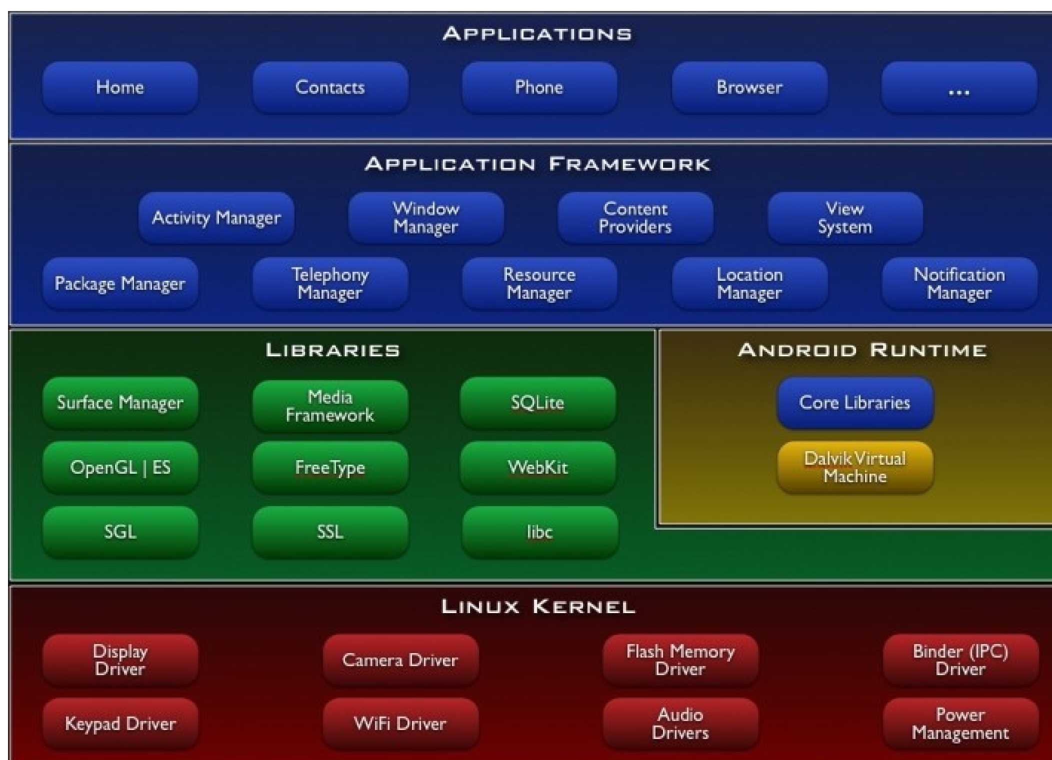

Figura 2.1 - Componentes do sistema operacional Android

Fonte: Google Code [5]

2.1.2.1 - Aplicativos

O *Android* já possui um conjunto de aplicativos, incluindo: cliente de email, programa para envio de SMS, calendário, mapas, navegador, agenda de contatos entre outros. Todos foram feitos utilizando a linguagem de programação Java.

2.1.2.2 - *Framework* dos Aplicativos

Os desenvolvedores têm acesso às mesmas APIs utilizadas pelas aplicações citadas acima. A arquitetura foi projetada para simplificar a reutilização de componentes, permitindo até a substituição de um aplicativo pelo usuário.

Existe um conjunto de serviços e sistemas que são utilizados ao escrever uma aplicação para o *Android*. Alguns desses componentes incluem:

- Um rico e extenso conjunto de *Views* que auxiliam a criação de uma aplicação possuindo listas, tabelas, caixas de texto, botões e até navegadores.

- *Content Providers* que permitem o compartilhamento de dados entre as aplicações.
- *Resource Manager* para acessar recursos como figuras e outros arquivos.
- *Notification Manager* que permite a exibição de alertas personalizados na barra de status.
- Activity *Manager* que gerencia o ciclo de vida das aplicações e permite a navegação entre elas.

2.1.2.3 - Bibliotecas

O *Android* possui um conjunto de bibliotecas C/C++ que é utilizado por vários componentes do sistema. Elas estão disponíveis para os desenvolvedores a partir do *framework*. Dentre as bibliotecas utilizadas, destacam-se:

- Biblioteca C do sistema: derivada da biblioteca padrão do sistema operacional BSD modificada para dispositivos baseados em Linux.
- Bibliotecas Multimídia: baseada na OpenCORE da PacketVideo. Suportam execução e gravação dos formatos de áudio e vídeo mais populares, assim como arquivos de imagem estática. Por exemplo: MPEG4, H.264, MP3, AAC, AMR, JPG e PNG.
- *Surface Manager*: Gráficos 2D e 3D a partir de várias aplicações.
- *LibWebCore*: um *engine* moderno para navegadores que gerencia tanto o navegador principal do *Android* quanto um navegador dentro de uma aplicação.
- SGL: *Engine* para gráficos.
- Bibliotecas 3D: uma implementação baseada nas APIs do OpenGL ES 1.0. Elas usam tanto a aceleração de *hardware* quando disponível, quanto um renderizador 3D otimizado por *software*.
- FreeType: renderização de *bitmaps* e fontes.
- SQLite: *engine* para bancos de dados relacionais. Disponível para todas as aplicações.

2.1.2.4 - *Android Runtime*

No *Android*, toda aplicação é executada em um processo próprio. Isso é possível pois cada aplicativo roda em uma instância própria da máquina virtual Dalvik. O Dalvik foi escrito de forma que várias instâncias da máquina virtual seja executada em um mesmo dispositivo de maneira eficiente.

A máquina virtual Dalvik executa arquivos do formato Dalvik Executable (.dex) que nada mais é do que classes java compiladas e otimizadas para um consumo baixo de memória. Além disso, ela invoca o *kernel* Linux para funcionalidades como *threading* e gerenciamento de memória de baixo nível.

2.1.2.5 - *Kernel* Linux

O *Android* utiliza a versão 2.6 de um *kernel* Linux para os serviços principais do sistemas, tais como: segurança, gerenciamento de memória, gerenciamento de processos, rede e *drivers*. Esse *kernel* também atua como uma camada abstrata entre o *hardware* e os outros *softwares*.

2.1.3 - Ambiente de Desenvolvimento

Essa seção descreve como baixar e instalar o SDK do *Android*, o *plugin* ADT e configurar o ambiente de desenvolvimento tornando possível a criação de aplicações no *Android*.

2.1.3.1 - Requisitos Mínimos

Para o desenvolvimento do 3DViewer, o sistema operacional utilizado foi o Windows XP e o ambiente de desenvolvimento foi o Eclipse Europa, por isso todas as configurações serão baseadas nesses componentes.

Abaixo está a lista completa do que pode ser utilizado para começar a construir aplicativos para o *Android*.

Os seguinte sistemas operacionais são oficialmente suportados:

- Windows XP ou Vista
- Mac OS X 10.4.8 ou superior (x86)

- Linux (Ubuntu Dapper Drake)

O ambiente de desenvolvimento Eclipse é fortemente recomendado por possuir componentes que se integram ao Android e facilitam o desenvolvimento. O Eclipse deve atender alguns requisitos para que essa integração seja possível. Ainda assim, outros ambientes podem ser utilizados para o desenvolvimento. Abaixo, consta uma lista dos requisitos de cada um dos possíveis ambientes:

- Eclipse IDE
  - ◇ Eclipse 3.3 (Europa) ou 3.4 (Ganymede)
    - ▪ Plugin Eclipse JDT (incluso na grande maioria dos pacotes do Eclipse)
    - ▪ WST (opcional)
  - ◇ JDK 5 ou JDK 6 (JRE não é suficiente)
  - ◇ *Plugin Android Development Tools* (opcional)
  - ◇ Não é compatível com o compilador GNU para Java (gcj)
- Outros ambientes de desenvolvimento ou IDEs
  - ◇ JDK 5 ou JDK 6 (JRE não é suficiente)
  - ◇ Apache Ant 1.6.5 ou superior para Linux e Mac, 1.7 ou superior para Windows
  - ◇ Não é compatível com o compilador GNU para Java (gcj)

2.1.3.2 - Instalação

Antes de instalar, é necessário baixar o SDK através do site http://code.google.com/android/download.html. O processo de instalação inclui apenas dois passos básicos: descompactar o conteúdo do arquivo baixado em um local do computador e adicionar o caminho da pasta tools à variável de ambiente PATH do sistema operacional. Isso permite que algumas ferramentas de linha de comando sejam utilizadas sem a necessidade de digitar o caminho completo de onde estão localizadas.

2.1.3.3 - *Android Development Tools Plugin*

Ao utilizar o Eclipse como ambiente de desenvolvimento, é possível instalar um *plugin* chamado *Android Development Tools*. Esse plugin permite a integração do Eclipse com as ferramentas e projetos do *Android*, facilitando e agilizando a criação, execução e busca por erros nas aplicações.

Os passos descritos na tabela 2.1 indicam como baixar e instalar o plugin ADT de acordo com a versão do Eclipse escolhida.

Eclipse 3.3 (Europa)	Eclipse 3.4 (Ganymede)
1. Abra o Eclipse e selecione *Help* > *Software Updates* > *Find and install...* 2. Na próxima janela, selecione *Search for new features to install* e clique em *Next*. 3. Clique em *New Remote Site.* 4. Na próxima janela, digite um nome	1. Abra o Eclipse e selecione Help > Software Updates... 2. Na próxima janela, clique na aba Available Software. 3. Clique em Add Site... 4. Digite em Location: https://dl-ssl.google.com/android/eclipse

para o site remoto (ex. Android Plugin) e digite a URL: https://dl-ssl.google.com/android/eclipse Clique em OK. 5. O novo site deve estar adicionado à lista de busca e selecionado. Clique em *Finish*. 6. Na janela *Search Results*, selecione *Android Plugin > Developer Tools*. Isso irá selecionar as ferramentas *Android Developer Tools* e *Android Editors*. Essa última é opcional, porém recomendada. Caso escolha instalá-la, é necessário o *plugin* WST conforme mencionado nos requisitos mínimos necessários. Clique em *Next*. 7. Selecione *Accept terms of the license agreement* e clique em *Next*. 8. Clique em *Finish*. 9. O plugin ADT não é assinado. Aceite a instalação clicando em *Install All*. 10. Reinicie o Eclipse.	Clique em OK. 5. Volte para a aba Avaiable Software, selecione Developer Tools e clique em Install... 6. Na janela seguinte, os itens “Android Developer Tools” e “Android Editors” devem estar selecionados. Esse último é opcional, porém recomendado. Caso escolha instalá-lo, é necessário o plugin WST conforme mencionado nos requisitos mínimos necessários. Clique em Finish. 7. Reinicie o Eclipse.

Tabela 2.1 - Instalando o Android Development Tools no Eclipse

Ao abrir o Eclipse novamente, é necessário atualizar o diretório do SDK nas preferências:

1. Selecione Window > Preferences... para abrir o painel de preferências.
2. Selecione Android na lista à esquerda.
3. Clique em Browse... e localize o diretório onde foi instalado o SDK.
4. Clique em Apply e depois em OK.

2.1.4 - Componentes de um Aplicativo no Android

Ao contrário do que é encontrado geralmente em outros sistemas, os aplicativos do Android não possuem um único ponto de entrada como uma função main(), por exemplo.

Eles possuem componentes essenciais que o sistema pode instanciar e rodar conforme a necessidade. Existem quatro tipos de componentes:

- Activity
- Service
- Broadcast Receiver
- Content Provider

Desses quatro componentes, apenas o primeiro é utilizado no 3DViewer. Uma *activity* apresenta a interface visual que um usuário deve ver sob um determinado aspecto no aplicativo. Ele pode ter uma *activity* principal e quantas *activities* secundárias precisar, de modo que o usuário navegue pelas *activities* existentes. Uma *activity* pode ter menus, listas, botões, figuras, etc. O conteúdo visual de uma *activity* é adicionado através de uma *view* que conceitualmente pode ser considerada a parte do aplicativo que faz a interação com o usuário. Tanto a *activity* quanto a *view* são classes existentes na API de desenvolvimento do Android e as suas funções ficarão mais claras através dos exemplos vistos daqui para frente. Uma explicação detalhada sobre cada um dos quatro componentes pode ser encontrada na página oficial do projeto Android.

Na figura 2.2 é possível ver o ciclo de vida de uma *activity* em um aplicativo do

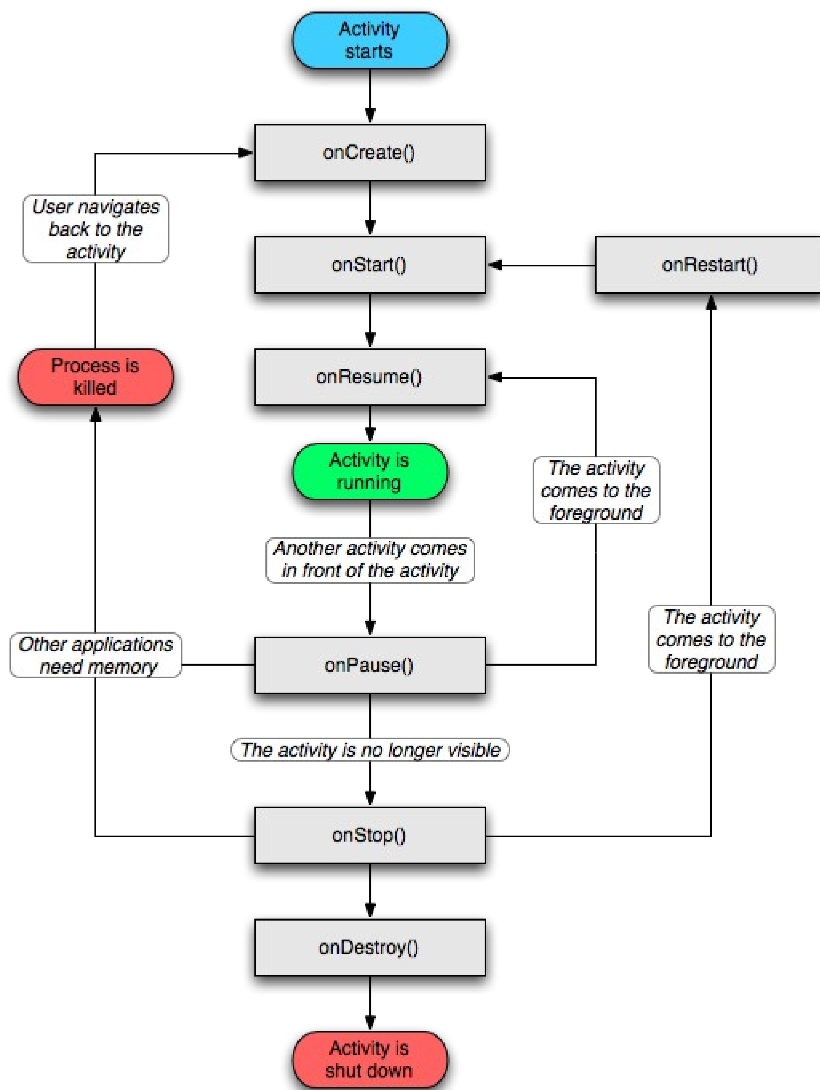

Figura 2.2 - Ciclo de vida de uma Activity

Fonte: Google Code [5]

Android.

2.1.5 - Construindo o Primeiro Aplicativo: "Hello, World"

A partir dos próximos passos será possível construir qualquer aplicativo para o Android. Nesse primeiro exemplo, é mostrado ainda a maneira correta de se criar um novo projeto, construir a interface do usuário, rodar o aplicativo e fazer o *debug* do mesmo no Eclipse.

2.1.5.1 - Criando um Novo Projeto

Após a instalação do SDK e do *plugin* para Eclipe, algumas funcionalidades são adicionadas ao ambiente de desenvolvimento. Para criar um projeto de um aplicativo Android no Eclipse, siga os seguintes passos:

- No menu do Eclipse, clique em File > New > Project. Caso o plugin tenha sido instalado corretamente, uma janela que se abre deve conter, dentre outras opções, uma pasta chamada Android que contém apenas uma entrada chamada Android Project. Selecione essa entrada e clique em Next.
- Na próxima janela é necessário preencher os detalhes do projeto. No caso desse primeiro aplicativo, um exemplo de preenchimento pode ser visto abaixo, ao terminar clique em Finish:
  - ◇ Project Name: HelloAndroid
  - ◇ Package Name: com.example.hello
  - ◇ Activity Name: HelloAndroid
  - ◇ Application Name: Hello, Android
- Após o plugin completar a criação do novo projeto, é possíver ver uma classe chamada HelloAndroid através do caminho HelloAndroid > src > com.example.hello. O código encontrado no arquivo deve estar assim:

```
package com.example.hello;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

2.1.5.2 - Construindo a Interface do Usuário

A partir do código gerado, é possível modificá-lo para satisfazer a necessidade do usuário. No exemplo, a frase "Hello, Android" deve ser mostrada na tela e para isso é preciso fazer a seguinte modificação:

```
package com.example.hello;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

Conforme dito anteriormente, a classe *HelloAndroid* é baseada na classe *Activity*. Uma *activity* nada mais é do que uma entrada no aplicativo que executa algumas ações específicas. Uma aplicação pode ter várias *Activities*, porém o usuário fará interação com uma de cada vez. Alem disso, na maioria das situações uma *activity* terá uma interface para o usuário, mas não é obrigatório que tenha.

Uma interface do usuário no Android é composta de hierarquias de objetos chamados de *Views*. Uma *view* é simplesmente um objeto de desenho como *radio buttons*, uma animação ou nesse caso, um texto. A subclasse utilizada no exemplo acima para escrever textos na tela é chamada de *TextView*.

2.1.5.3 - Executando o Aplicativo

O *plugin* para Eclipse torna a execução de qualquer aplicativo em algo bem simples. No menu, selecione Run > Open Run Dialog, depois clique duas vezes em Android Application. Uma nova configuração será criada com o nome New_configuration. Mude o nome para algo que faça sentido, como "Hello, Android" e

escolha o projeto criado clicando no botão Browse. Para finalmente, executar o aplicativo, clique em Run. O emulador será carregado e ao final do *boot*, o aplicativo "Hello, Android" aparecerá na tela.

2.1.5.4 - Utilizando um Layout XML na UI

Como em qualquer outra linguagem, construir a interface do usuário através de códigos pode ser tornar uma tarefa complexa e trabalhosa. Para amenizar essa dificuldade é possível criar essas interfaces através de arquivos XML. Utilizando o plugin do Android para Eclipse, o desenvolvedor é capaz de construir a interface através de um sistema WYSIWYG.

A estrutura do arquivo XML utilizado é semelhante à da classe View junto com os seus atributos e métodos. Porém a nomenclatura nem sempre é igual nos dois casos e por isso a leitura da documentação do Android é altamente recomendada.

2.1.5.5 - Utilizando o *Debugger* no Projeto

O *plugin* do Android tem uma ótima integração com o Eclipse. Qualquer erro de código pode ser rastreado utilizando o *debugger* em conjunto com *breakpoints* e a perspectiva de *debug* do Eclipse. Esse é processo seguido geralmente em qualquer aplicativo java.

2.2 - OpenGL

2.2.1 - Introdução

Segundo Isabel Manssour [1], OpenGL pode ser definida como uma biblioteca de rotinas gráficas e de modelagem bidimensional e tridimensional. A maior vantagem na sua utilização é a sua extrema rapidez, no entanto, o fato de ser altamente portável também merece destaque.

Ao contrário do que possa parecer inicialmente, o OpenGL não é uma linguagem de programação. Trata-se, na verdade, de uma poderosa e sofisticada API para criação de aplicações gráficas. Portanto, diz-se que um programa é baseado em OpenGL ou é uma aplicação OpenGL quando é escrito em alguma linguagem de programação que faz chamadas a uma ou mais bibliotecas OpenGL.

Basicamente, a biblioteca OpenGL permite a partir de funções próprias o desenho de primitivas gráficas, tais como linhas e polígonos. Porém, ela dá suporte a alguns efeitos como iluminação, colorização, mapeamento de textura, transparência, animação, entre outros. É por tudo isso que, atualmente, "OpenGL é reconhecida e aceita como um padrão API para desenvolvimento de aplicações gráficas 3D em tempo real" [1].

2.2.2 - OpenGL ES

O Android inclui em sua API, suporte para gráficos tridimensionais de alta performance através do OpenGL.

Na verdade, o que se tem no Android é apenas um tipo de API OpenGL chamada OpenGL ES que foi desenvolvida especialmente para dispositivos com sistemas embarcados como celulares e veículos.

As versões do OpenGL ES são, geralmente, bastante semelhantes à biblioteca padrão utilizada em desktops e laptops. Especificamente no Android, a versão utilizada é a 1.0 que corresponde à versão 1.3 do OpenGL. Portanto, é de se esperar que um software feito com base na especificação 1.3 do OpenGL padrão, funcione ao ser migrado para o Android.

2.2.3 - Funcionamento Básico

É possível renderizar imagens bidimensionais e tridimensionais através do OpenGL. Contudo, o processo para gerar imagens tridimensionais pode parecer complicado por conta do poder que a API disponibiliza ao desenvolvedor.

Nesta seção, é explicado os passos a serem seguidos para a renderização de objetos tridimensionais utilizando OpenGL. O processo é análogo a tirar uma fotografia com uma máquina fotográfica que segue os passos abaixo (Manssour [1] apud Woo [2]):

- Arrumar o tripé e posicionar a câmera para fotografar a cena. Equivalente a especificar as transformações de visualização.
- Arrumar a cena para ser fotografada, incluindo ou excluindo objetos e pessoas. Equivalente à etapa de modelagem que inclui as transformações geométricas e o desenho dos objetos.

- Escolher a lente da câmera ou ajustar o zoom. Equivalente a especificar as transformações de projeção.
- Determinar o tamanho final da foto (maior ou menor). Equivalente a especificar a *viewport* que pode ser entendido como a janela de visão da cena.

Esses passos tem por objetivo traçar uma direção a ser seguida antes de começar a desenvolver um software em OpenGL. Apesar disso, em uma situação específica uma etapa pode ser pulada ou modificada de modo que facilite o desenvolvimento. O *zoom*, por exemplo, através dos passos, consta como uma etapa relacionada a transformação de projeção. Logo, estaria ligado diretamente à câmera fotográfica em uma fotografia. Porém, em alguns casos, poderia ser melhor atribuir essa etapa à fase de arrumar a cena.

Certamente, ao tirar fotos, o melhor a fazer é ajustar o zoom através da câmera, já que aproximar ou afastar a cena pode ser uma tarefa difícil. Mas fazer isso em OpenGL, deixando a projeção e a janela de visão sempre iguais, pode trazer alguns benefícios.

De uma forma simplificada, é de se esperar que ao trabalhar com OpenGL o desenvolvedor fixe os objetos ou a visão perspectiva do usuário e trabalhe com as transformações em apenas uma dessas duas alternativas. Caso as transformações sejam aplicadas em ambas, o *software* pode se tornar confuso e extremamente complexo.

2.3 - Arquivos DXF

O formato DXF é uma representação de dados através de *tags* criado para descrever todo tipo de informação contida em um arquivo de desenho do software AutoCAD.

Cada dado no arquivo DXF é precedido de um número inteiro chamado de *group code*. O valor do *group code* indica qual o tipo de dado vem a seguir e qual é o significado desse dado para um determinado objeto.

Os elementos dentro do arquivo estão divididos em seções. Abaixo estão especificadas todas as possíveis seções existentes em um arquivo DXF:

- *HEADER*
- *CLASSES*
- *TABLES*

- *BLOCKS*
- *ENTITIES*
- *OBJECTS*
- *THUMBNAILIMAGE*

Na seção *Tables* podem ser definidos objetos chamados de LAYER. Cada *layer* possui um nome e uma cor. Eles serão utilizados no 3DViewer apenas quando um objeto gráfico não possuir uma cor definida. Caso seja especificada uma cor, a cor do *layer* é ignorada e a definição de um *layer* passa a não fazer sentido. O *layer* pode ser utilizado para casos em que vários objetos gráficos possuem a mesma cor.

A seção que representa os objetos gráficos do desenho é a *Entities*. Por esse motivo e também por questões de simplicidade, apenas essa seção é realmente utilizada pelo 3DViewer. Todas as outras seções são ignoradas e podem ou não existir caso o arquivo seja lido somente pelo 3DViewer. É de se esperar também que nem todo arquivo DXF consiga ser aberto pelo 3DViewer por conta desse limitação.

A seção *Entities* possui uma vasta coleção de objetos gráficos que podem ser representados. Porém, o 3DViewer trata apenas dois deles: 3DFACE e LINE. Com esses elementos, é possível representar triângulos e quadriláteros ou linhas, respectivamente. Novamente, visando a simplicidade, o 3DViewer sempre considera o 3DFACE como sendo um triângulo.

Na figura 2.3 é possível ver uma parte de um arquivo DXF. Essa parte é suficiente para que o 3DViewer consiga renderizar a imagem, porém pode não ser possível fazer o mesmo em outros softwares que exijam a presença de outras seções do formato DXF. Os campos destacados são os *group codes* mencionados anteriormente.

```
0
SECTION
2
ENTITIES
0
3DFACE
8
1
62
16
10
-1.0
20
1.0
30
1.0
11
1.0
21
1.0
31
1.0
12
-1.0
22
-1.0
32
1.0
13
-1.0
23
-1.0
33
1.0
0
ENDSEC
0
EOF
```

Figura 2.3 - Exemplo de parte de um arquivo DXF.

Como pode ser observado, o 3DFACE é especificado basicamente por três *group codes* que definem sua cor, as coordenadas de cada vértice e o *layer* ao qual pertence. O *group code* igual a 8 indica que o próximo dado será o *layer* e o *group code* igual a 62 indica qual será a cor no formato ACI. Os conjuntos de *group codes* 10, 20, 30; 11, 21, 31; 12, 22, 32 e 13, 23, 33 indicam, respectivamente, as coordenadas x, y e z de cada um dos quatro vértices.

A documentação do formato DXF indica que se os *group codes* 12, 22 e 32 forem iguais aos 13, 23 e 33 dentro do 3DFACE , então trata-se de um triângulo, caso contrário trata-se de um quadrilátero. Porém, conforme dito anteriormente, o 3DViewer sempre vai considerá-los triângulos, ignorando as coordenadas relativas ao quarto vértice.

Analogamente, uma linha pode ser definida em um arquivo DXF utilizando a *string* LINE e os conjuntos de *group codes* 10, 20, 30 e 11, 21, 31 para especificar as coordenadas x, y, z dos pontos inicial e final, respectivamente, da reta.

É importante dizer que o formato DXF ainda é bastante utilizado por desenvolvedores ligados à computação gráfica, principalmente pelo legado deixado durante anos, porém conforme o AutoCAD fica mais poderoso e utiliza cada vez mais tipos de objetos complexos, esse formato vai se tornando menos usual. Além disso, alguns objetos não são suficientemente documentados para que possam ser utilizados.

Capítulo 3
O 3DViewer

3.1 - Introdução

Softwares que utilizam o OpenGL como API gráfica já não são novidade nos computadores convencionais. Porém, o uso dessa tecnologia em dispositivos móveis ainda não é tão difundida assim. Por isso, a idéia de fazer o 3DViewer surgiu.

Além disso, o fato de poder testar o novo sistema operacional criado pela Google, contribuiu para que a motivação fosse ainda maior.

Diante da grande quantidade de tipos de softwares que utilizam renderização de imagens tridimensionais, era necessário escolher um para desenvolver. E a escolha por um visualizador não foi por acaso.

O 3DViewer foi baseado em um software de exemplo que acompanha a biblioteca *wxWidgets*. Ela é uma biblioteca multiplataforma que disponibiliza uma API usada para criação de programas GUI.

Ao fazer o 3DViewer, seria possível demonstrar algumas funcionalidades presentes no Android como leitura de arquivos, criação de uma UI e renderização de imagens tridimensionais utilizando OpenGL. Assim como o *software* de exemplo demonstra algumas ferramentas disponíveis na biblioteca *wxWidgets*.

Na figura 3.1, é possível ver o software de demonstração que acompanha a biblioteca *wxWidgets*.

Figura 3.1 - Software de exemplo da biblioteca wxWidgets

3.2 - A Solução Escolhida

Após a decisão sobre o que seria feito, a fase de desenvolvimento só poderia começar depois que fosse resolvido como o 3DViewer seria construido. A idéia inicial era aproveitar o código fonte do software de demonstração e transportá-lo para dentro do Android.

A biblioteca *wxWidgets* suporta várias linguagens de programação incluindo C++ e conforme dito no capítulo 2, o Android possui uma vasta biblioteca C/C++ disponível e além disso, foi feito baseado em um *kernel* linux bem atualizado.

Portanto, os passos para que o 3DViewer fosse desenvolvido, poderiam ser resumidos dessa forma:

- Selecionar os códigos-fonte necessários para compilar o software de exemplo da wxWidgets.
- Baixar e compilar a biblioteca wxWidgets dentro do *kernel* linux do Android.
- Compilar os fontes para gerar o 3DViewer fazendo as adaptações necessárias.

Porém, após muita pesquisa, foi constatado que não seria possível compilar a biblioteca wxWidgets dentro do *kernel* linux do Android. Isso acontece porque não há

compilador ou qualquer outra ferramenta de desenvolvimento dentro do linux no qual o Android é baseado.

Além disso, a interface disponibilizada para o acesso ao linux é muito pobre. Trata-se de um executável que emula o ambiente através da linha de comando do sistema operacional utilizado. Nesse ambiente emulado, nem todos os comandos básicos do linux estão disponíveis.

Acessando a pasta *tools* dentro da pasta raiz do SDK do Android, é possível ver alguns executáveis. O programa emulator.exe abre um emulador de celular rodando o Android e o adb.exe é usado para fazer o acesso ao linux. Após abrir o emulador, o linux é acessado através da linha de comando conforme mostra a figura 3.2.

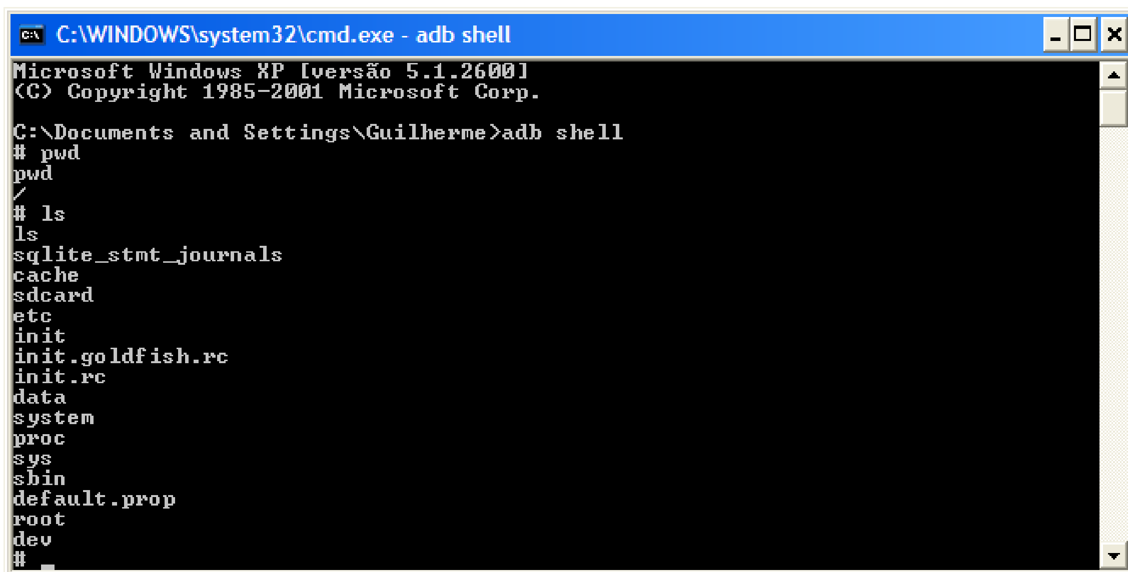

Figura 3.2 - Acesso ao linux emulado do Android

É importante lembrar que nesse momento a idéia não era utilizar o SDK disponibilizado pela Google e sim aproveitar o fato de que o Android era executado sob um linux e que por isso poderia rodar aplicativos feitos em linguagem nativa.

Então, para que fosse possível compilar tanto a biblioteca wxWidgets quanto o que seria o 3DViewer feito em C++, era necessário um ambiente de desenvolvimento específico para C++ e que contemplasse os sistemas embarcados.

O ambiente de desenvolvimento encontrado foi o Sourcery G++. Esse ambiente foi desenvolvido pela empresa CodeSorcery, cuja atividade principal é a criação de ferramentas para desenvolvimento em dispositivos embarcados.

A edição *lite* do Sourcery G++ é gratuita, conta com compiladores GNU C/C++ e possui uma versão específica para os processadores ARM.

O Android foi concebido para rodar utilizando esses processadores. Eles são muito comuns em dispositivos móveis e portanto, as instruções dos programas compilados através do Sourcery G++ deveriam ser compatíveis com o Android.

A partir daí, o próximo passo era tentar executar o programa mais simples possível em C++ no Android, porém compilado fora dele. Através do adb.exe também era possível importar e exportar arquivos do ambiente linux. Sabendo disso, compilou-se o “Hello, World” em C++ utilizando o Sourcery G++. Após a importação do binário para o linux, o programa foi executado com sucesso imprimindo uma frase na linha de comando emulada.

O passo seguinte seria a compilação da biblioteca wxWidgets e consequentemente do 3DViewer. Porém, a tentativa de compilação gerou vários erros de dependência de outras bibliotecas, inclusive de bibliotecas internas do linux mas que não foram encontradas pelo Sourcery G++.

Consultando a comunidade de desenvolvimento envolvida com o Android, foi possível constatar a insatisfação de alguns desenvolvedores por não poderem reaproveitar seus códigos pelo fato da Google não disponibilizar um SDK para criação de aplicativos em C++.

Em alguns casos, os desenvolvedores compilavam biblioteca a biblioteca para que seus programas pudessem funcionar no Android. Ainda assim, era necessário o uso de JNI para que eles tivessem uma interface gráfica, já que não havia outro modo de executar esses programas que não fosse pela linha de comando.

O JNI é um *framework* que permite a chamada de códigos em linguagem nativa como C, C++ e *assembly* através de um *software* feito em Java. Assim, os acessos a classes, métodos e atributos feitos em C++ eram feitos a partir do código Java gerado pelo único SDK lançado pela Google.

Com isso, optou-se por fazer o 3DViewer inteiramente em Java e utilizando o SDK do Android juntamente com o *plugin* do Eclipse. Caso contrário, o desenvolvimento e a procura por erros se tornariam extremamente complexos. Aliado a isso, o tempo de desenvolvimento do 3DViewer estipulado inicialmente ficaria comprometido.

3.3 - Desenvolvimento Inicial

Em um primeiro momento, o foco era fazer com que um arquivo lido pelo 3DViewer fosse desenhado na tela. Para isso, era necessário transferir um arquivo no formato DXF para dentro do Android, programar o 3DViewer para que buscasse esse arquivo em um local pré-definido e a partir dos seus dados renderizasse os triângulos ou linhas usando OpenGL.

A transferência do arquivo pode ser feita de duas formas. A primeira é através da linha de comando utilizando o executável adb.exe. Os comandos “adb push” e “adb pull”, importam e exportam, respectivamente, arquivos para o Android. A segunda maneira é através do Eclipse, utilizando uma nova perspectiva chamada DDMS criada pelo *plugin* do Android, conforme mostra a figura 3.3.

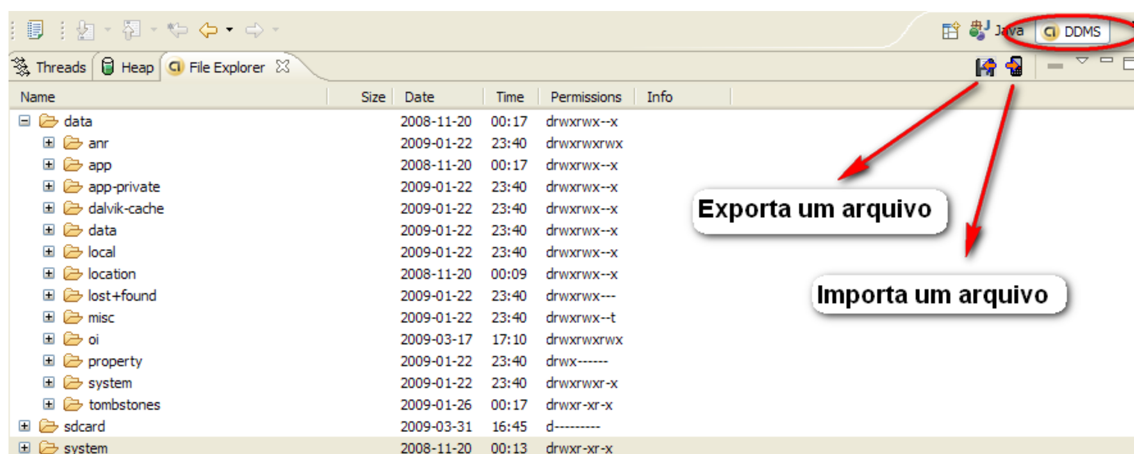

Figura 3.3 - Importar ou exportar um arquivo para o emulador do Android através do Eclipse

Para ter acesso ao arquivo, basta utilizar os *packages* básicos de IO existentes na linguagem Java. É preciso ter atenção, pois cada software só tem acesso aos diretórios pertencentes a ele. Além disso, cada emulador aberto é uma instância independente e não compartilha os mesmos arquivos e diretórios criados.

Os dois métodos de importação de arquivos permitem que os arquivos sejam colocados em qualquer pasta do sistema, porém qualquer tentativa de leitura através do software de uma pasta que não lhe pertence não irá gerar qualquer tipo de exceção.

Uma terceira alternativa adotada durante parte do desenvolvimento foi incorporar o arquivo DXF como parte do programa e acessá-lo pela classe R criada, por padrão, em qualquer projeto do Android.

A classe R não deve ser editada manualmente e serve para acessar recursos que não fazem parte diretamente do programa criado. Ela é adicionada assim que um novo projeto é criado no Eclipse e está sempre sincronizada com o conteúdo do diretório res, conforme mostra a figura 3.4.

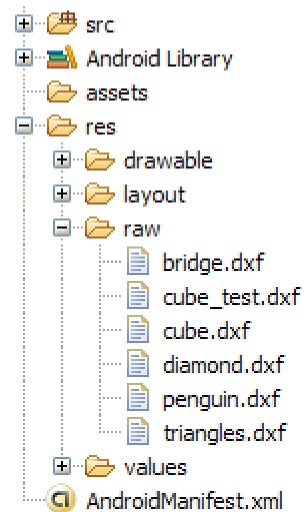

Figura 3.4 - Estrutura de um projeto do Android no Eclipse

De acordo com a figura 3.4, supondo que existisse somente arquivos na pasta raw, a classe R teria o seguinte formato:

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
    }
    public static final class layout {
    }
    public static final class raw {
        public static final int bridge=0x7f040000;
        public static final int cube=0x7f040001;
        public static final int cube_test=0x7f040002;
        public static final int diamond=0x7f040003;
        public static final int penguin=0x7f040004;
        public static final int triangles=0x7f040005;
    }
    public static final class string {
    }
}
```

As classes *string* e *attr* apontam para os valores encontrados em um arquivo XML dentro da pasta values. A classe *drawable* aponta para imagens utilizadas no programa e que ficam dentro da pasta drawable. A classe *layout* contém constantes que apontam para as *views* criadas através da interface WYSIWYG do Eclipse e por fim, a classe *raw* contém os valores para os arquivos colocados na pasta raw.

As constantes criadas representam um ID para cada objeto e podem ser acessados em qualquer lugar do programa através da classe R. Por exemplo, para acessar o ID do arquivo bridge.dxf, basta digitar `R.raw.bridge`.

Com o arquivo DXF dentro do Android, a próxima etapa era ler o arquivo identificando cada *group code* e armazenar os valores encontrados. Para isso, foi feito o método *load* que recebe um *stream* do arquivo DXF e começa a percorrer suas linhas. Basicamente, o método procura pelos group codes de início de seção e fim de arquivo e chama um outro método específico para cada uma das seções encontradas, como pode ser visto no trecho de código abaixo.

```
(...)
for (line1 = GetLines(text), line2 = GetLines(text);
     line2.compareTo("EOF") != 0;
     line1 = GetLines(text), line2 = GetLines(text)) {
    if (line1.compareTo("999") == 0)
        continue;
    else if ((line1.compareTo("0")==0)&&(line2.compareTo("SECTION")==0)) {
        line1 = GetLines(text);
        line2 = GetLines(text);
        if (line1.compareTo("2") == 0) {
            if (line2.compareTo("HEADER") == 0) {
                if (!ParseHeader(text))
                  return false;
            }
            else if (line2.compareTo("TABLES") == 0) {
                if (!ParseTables(text))
                    return false;
            }
            else if (line2.compareTo("ENTITIES") == 0) {
                if (!ParseEntities(text))
                    return false;
            }
        }
    }
}
(...)
```

O método *GetLines* retorna uma linha do arquivo e incrementa um contador global indicando a linha atual. O *group code* igual a 999 é considerado comentário no arquivo DXF.

Os três métodos que realizam o *parser* das seções *header*, *tables* e *entities*, seguem o mesmo algoritmo usado do método *load*. No entanto, o método *ParseHeader* apenas verifica se a seção *header* foi iniciada e finalizada corretamente, ignorando o seu conteúdo. Já os outros dois métodos guardam temporariamente os dados para que a imagem seja renderizada posteriormente.

```
boolean ParseHeader(BufferedReader text) {
    String line1 = null, line2 = null;
    for (line1 = GetLines(text), line2 =GetLines(text);
         line2.compareTo("EOF") != 0;
         line1 = GetLines(text), line2 = GetLines(text)) {
        if ((line1.compareTo("0") == 0) && (line2.compareTo("ENDSEC") == 0))
            return true;
    }
    return false;
}
```

O método *ParseTables* procura por objetos do tipo *layer* no arquivo e ao achar uma ocorrência, seu nome e cor são colocados em um *arraylist* de *layers*.

```
(...)
for (line1 = GetLines(text), line2 = GetLines(text);
     line2.compareTo("EOF") != 0;
     line1 = GetLines(text), line2 = GetLines(text)) {
    if ((line1.compareTo("0") == 0) && inlayer) {
        if ((layer.name != null) && (layer.colour != -1)) {
            DxfLayer p = new DxfLayer();
            p.name = layer.name;
            p.colour = layer.colour;
            mLayers.add(p);
        }
        layer = new DxfLayer();
        inlayer = false;
    }
    if ((line1.compareTo("0") == 0) && (line2.compareTo("ENDSEC") == 0))
        return true;
    else if ((line1.compareTo("0")==0) && (line2.compareTo("LAYER")==0))
        inlayer = true;
    else if (inlayer) {
        if (line1.compareTo("2") == 0) // layer name
```

```
            layer.name = line2;
        else if (line1.compareTo("62") == 0) { // ACI colour
            layer.colour = (int) Long.parseLong(line2);
        }
    }
}
(...)
```

Do mesmo modo, o método *ParseEntities* faz a busca por elementos gráficos do tipo *line* e *3dface* e guarda suas informações em um *arraylist* de *entities*.

```
(...)
for (line1 = GetLines(text), line2 = GetLines(text);
     line2.compareTo("EOF") != 0;
     line1 = GetLines(text), line2 = GetLines(text)) {
    if ((line1.compareTo("0") == 0) && state > 0) {
        if (state == 1) { // 3DFACE
            DxfFace p = new DxfFace();
            p.v0 = new DxfVector(v[0].x, v[0].y, v[0].z);
            p.v1 = new DxfVector(v[1].x, v[1].y, v[1].z);
            p.v2 = new DxfVector(v[2].x, v[2].y, v[2].z);
            p.v3 = new DxfVector(v[3].x, v[3].y, v[3].z);
            p.CalculateNormal();
            if (colour != -1)
                p.colour = colour;
            else
                p.colour = GetLayerColour(layer);
            mEntities.add(p);
            colour = -1;
            layer = null;
            for (int i = 0; i < v.length; i++)
             v[i] = new DxfVector();
            state = 0;
        }
        else if (state == 2) { // LINE
            DxfLine p = new DxfLine();
            p.v0 = new DxfVector(v[0].x, v[0].y, v[0].z);
            p.v1 = new DxfVector(v[1].x, v[1].y, v[1].z);
            if (colour != -1)
                p.colour = colour;
            else
                p.colour = GetLayerColour(layer);
            mEntities.add(p);
            colour = -1;
            layer = null;
```

```
                for (int i = 0; i < v.length; i++)
                 v[i] = new DxfVector();
                state = 0;
            }
        }
        if ((line1.compareTo("0") == 0) && (line2.compareTo("ENDSEC") == 0))
            return true;
        else if ((line1.compareTo("0") == 0) && (line2.compareTo("3DFACE") == 0))
            state = 1;
        else if ((line1.compareTo("0") == 0) && (line2.compareTo("LINE") == 0))
            state = 2;
        else if (state > 0) {
            double d = Double.parseDouble(line2);
            if (line1.compareTo("10") == 0)
                v[0].x = (float) d;
            else if (line1.compareTo("20") == 0)
                v[0].y = (float) d;
            else if (line1.compareTo("30") == 0)
                v[0].z = (float) d;
            else if (line1.compareTo("11") == 0)
                v[1].x = (float) d;
            else if (line1.compareTo("21") == 0)
                v[1].y = (float) d;
            else if (line1.compareTo("31") == 0)
                v[1].z = (float) d;
            else if (line1.compareTo("12") == 0)
                v[2].x = (float) d;
            else if (line1.compareTo("22") == 0)
                v[2].y = (float) d;
            else if (line1.compareTo("32") == 0)
                v[2].z = (float) d;
            else if (line1.compareTo("13") == 0)
                v[3].x = (float) d;
            else if (line1.compareTo("23") == 0)
                v[3].y = (float) d;
            else if (line1.compareTo("33") == 0)
                v[3].z = (float) d;
            else if (line1.compareTo("8") == 0)  // layer
                layer = line2;
            else if (line1.compareTo("62") == 0) // colour
                colour = (int) Long.parseLong(line2);
        }
    }
(...)
```

Os elementos gráficos *3dface* e *line* são representados no 3DViewer como sendo objetos da classe *DxfFace* e *DxfLine*, respectivamente. Ambas as classes estendem a classe base *DxfEntity* que possui apenas dois atributos indicando o tipo de entidade (*line* ou *face*) e a sua cor.

A classe *DxfFace* possui cinco atributos. Eles são os vetores que representam o *3dface* lido do arquivo DXF. Além dos quatro vetores lidos do arquivo, um quinto vetor é calculado através do método *CalculateNormal* indicando o vetor normal ao triângulo.

A classe *DxfLine* possui apenas dois atributos representando os pontos inicial e final da linha que se quer desenhar.

Com todas essas informações, o arquivo DXF foi totalmente representado dentro do 3DViewer e portanto, não é mais necessário. O próximo passo consistia em ler cada elemento do *arraylist* de *entities* e renderizá-los utilizando a API do OpenGL ES.

Como os elementos *3dface* e *line* não possuem os mesmos atributos foi preciso separar o modo de se fazer a renderização por tipo de *entity*. Por isso a classe *DxfEntity* possui um atributo que indica qual o tipo de *entity* o objeto está representando. Assim, o método *render* foi criado. Parte dele pode ser visto abaixo.

```
(...)
for (DXFEntity p : mEntities) {
    int c = ACIColourToRGB(p.colour);
    if (p.type == DXFEntityType.Line) {
        DXFLine line = (DXFLine) p;
        gl.glColor4f(Color.red(c)/255.0f,
                     Color.green(c)/255.0f,
                     Color.blue(c)/255.0f, 1.0f);

        FloatBuffer fb = makeFloatBuffer(
                    new float[] { line.v0.x, line.v0.y, line.v0.z,
                                  line.v1.x, line.v1.y, line.v1.z });

        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, fb);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDrawArrays(GL10.GL_LINES, 0, 2);
    }
    else if (p.type == DXFEntityType.Face) {
        DXFFace face = (DXFFace) p;

        gl.glColor4f(Color.red(c)/255.0f,
                     Color.green(c)/255.0f,
                     Color.blue(c)/255.0f, 1.0f);
```

```
        gl.glNormal3f(face.n.x, face.n.y, face.n.z);

        FloatBuffer fb = makeFloatBuffer(
                    new float[] { face.v0.x, face.v0.y, face.v0.z,
                                  face.v1.x, face.v1.y, face.v1.z,
                                  face.v2.x, face.v2.y, face.v2.z });
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, fb);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    }
}
(...)
```

O algoritmo apresentado, apesar de funcionar, não mostrava a imagem de maneira satisfatória. Percorrer o *arraylist* de *entities* e renderizar os elementos um a um era muito custoso computacionalmente. A necessidade de identificar o tipo de *entity* lida e converter suas coordenadas dos vértices em um *buffer* de coordenadas para serem utilizadas nos métodos do OpenGL, deixava o software lento ao realizar transformações.

Para minimizar essa lentidão, o método *render* foi substituido pelo método *renderToBuffer*. Esse novo método separa os elementos do *arraylist* de *entities* em outros quatro *arraylists*. Um *arraylist* somente de cores, outro de vetores normais e mais dois de vetores que representam triângulos e linhas. Desse modo, o tempo perdido na identificação das *entities* e na criação dos *buffers* ocorreria apenas uma vez e antes da renderização da imagem.

O algoritmo do método *renderToBuffer* pode ser visto abaixo.

```
(...)
for (DxfEntity p : mEntities) {
    int c = ACIColourToRGB(p.colour);
    if (p.type == DxfEntityType.Line) {
        DxfLine line = (DxfLine) p;
        mColorsVertex.add(Color.red(c)/255.0f);
        mColorsVertex.add(Color.green(c)/255.0f);
        mColorsVertex.add(Color.blue(c)/255.0f);
        mColorsVertex.add(1.0f);
        mColorsVertex.add(Color.red(c)/255.0f);
        mColorsVertex.add(Color.green(c)/255.0f);
        mColorsVertex.add(Color.blue(c)/255.0f);
        mColorsVertex.add(1.0f);
```

```
            mLinesVertex.add(line.v0.x);
            mLinesVertex.add(line.v0.y);
            mLinesVertex.add(line.v0.z);
            mLinesVertex.add(line.v1.x);
            mLinesVertex.add(line.v1.y);
            mLinesVertex.add(line.v1.z);
        }
        else if (p.type == DxfEntityType.Face) {
            DxfFace face = (DxfFace) p;
            mColorsVertex.add(Color.red(c)/255.0f);
            mColorsVertex.add(Color.green(c)/255.0f);
            mColorsVertex.add(Color.blue(c)/255.0f);
            mColorsVertex.add(1.0f);
            mColorsVertex.add(Color.red(c)/255.0f);
            mColorsVertex.add(Color.green(c)/255.0f);
            mColorsVertex.add(Color.blue(c)/255.0f);
            mColorsVertex.add(1.0f);
            mColorsVertex.add(Color.red(c)/255.0f);
            mColorsVertex.add(Color.green(c)/255.0f);
            mColorsVertex.add(Color.blue(c)/255.0f);
            mColorsVertex.add(1.0f);

            mNormalsVertex.add(face.n.x);
            mNormalsVertex.add(face.n.y);
            mNormalsVertex.add(face.n.z);
            mNormalsVertex.add(face.n.x);
            mNormalsVertex.add(face.n.y);
            mNormalsVertex.add(face.n.z);
            mNormalsVertex.add(face.n.x);
            mNormalsVertex.add(face.n.y);
            mNormalsVertex.add(face.n.z);

            mFacesVertex.add(face.v0.x);
            mFacesVertex.add(face.v0.y);
            mFacesVertex.add(face.v0.z);
            mFacesVertex.add(face.v1.x);
            mFacesVertex.add(face.v1.y);
            mFacesVertex.add(face.v1.z);
            mFacesVertex.add(face.v2.x);
            mFacesVertex.add(face.v2.y);
            mFacesVertex.add(face.v2.z);
        }
    }
    (...)
```

Olhando para o código, é possível observar de forma bastante clara que para renderizar um triângulo são necessárias: as coordenadas de seus três vértices, o vetor normal a esse triângulo e sua cor. Porém, é de se estranhar que cada cor e cada vetor normal tenha sido adicionado três vezes seguidas para cada triângulo. Isso ocorre porque os três *arraylists* devem ter o mesmo número de elementos. No caso da cor, cada índice é formado por quatro *floats* representando a cor no formato RGBA, já para os vetores normais e triângulos, cada índice é o conjunto das coordenadas x, y e z.

Nesse momento, faltavam apenas a renderização através da API do OpenGL ES e a configuração do modo de visualização da imagem. Porém, antes disso, é importante ressaltar que os *arraylists* não são diretamente utilizados nos métodos de OpenGL. Eles são transformados em *buffers* de valores *float* através de métodos auxiliares.

Após essa conversão, a imagem é renderizada conforme o trecho de código abaixo.

```
(...)
// Draw loaded DXF
if (mColorBuffer != null) {
	gl.glColorPointer(4, GL10.GL_FLOAT, 0, mColorBuffer);
	gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
}

if (mNormalBuffer != null) {
	gl.glNormalPointer(GL10.GL_FLOAT, 0, mNormalBuffer);
	gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
}

if (mFaceBuffer != null) {
	gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFaceBuffer);
	gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
	gl.glDrawArrays(GL10.GL_TRIANGLES, 0, mFaceBuffer.capacity() / 3);
}

if (mLineBuffer != null) {
	gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mLineBuffer);
	gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
	gl.glDrawArrays(GL10.GL_LINES, 0, mLineBuffer.capacity() / 2);
}
(...)
```

Os métodos *glColorPointer*, *glNormalPointer* e *glVertexPointer* especificam os *buffers*, o tipo de dado contido nele e a quantidade de valores que foram utilizados para

representar, respectivamente, as cores, os vetores normais e as coordenadas de triângulos ou linhas. E é através do método *glDrawArrays* que cada um dos valores lidos desses *buffers* geram os triângulos e linhas lidas do arquivo DXF. O método *glEnableClientState* habilita o uso de cada um dos buffers.

Por fim, era preciso ajustar a janela de visão e a projeção desejada para que a visualização da imagem ficasse perfeita. Abaixo está o código utilizado para fazer essa configuração.

```
(...)
gl.glMatrixMode(GL10.GL_PROJECTION);
gl.glLoadIdentity();
gl.glViewport(0,0,w,h);
GLU.gluPerspective(gl, 45.0f, ((float)w)/h, 1.0f, 100.0f);
gl.glMatrixMode(GL10.GL_MODELVIEW);
gl.glLoadIdentity();
(...)
```

O método *glMatrixMode* especifica se os comandos a seguir serão com relação à projeção (GL_PROJECTION) ou aos objetos (GL_MODELVIEW). Antes de começar qualquer configuração ou transformação utiliza-se o método *glLoadIdentify*. Ele carrega a matriz identidade inibindo qualquer alteração feita anteriormente. Isso é necessário pois as transformações em OpenGL são acumulativas.

A janela de visão é configurada através de quatro parâmetros que especificam a coordenada do seu vértice inferior esquerdo, sua altura e seu comprimento. Geralmente o que se quer é a janela de visão igual ao tamanho da janela do aplicativo.

O método *gluPerspective* é usado para configurar a projeção perspectiva conforme mostra a figura 3.5. Através dele é possível especificar o ângulo do campo de visão na direção y em graus, a razão que resulta no campo de visão na direção x e as distâncias do observador com relação ao maior e ao menor plano de corte na direção z.

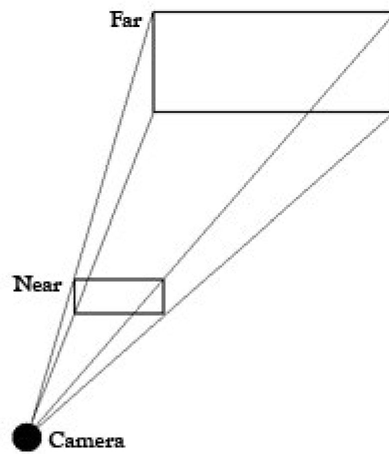

Figura 3.5 - Configuração da projeção perspectiva

Fonte: Zeus CMD [7]

No final dessa primeira etapa, o 3DViewer ao ser executado apresentava uma tela ao usuário semelhante à mostrada na figura 3.6.

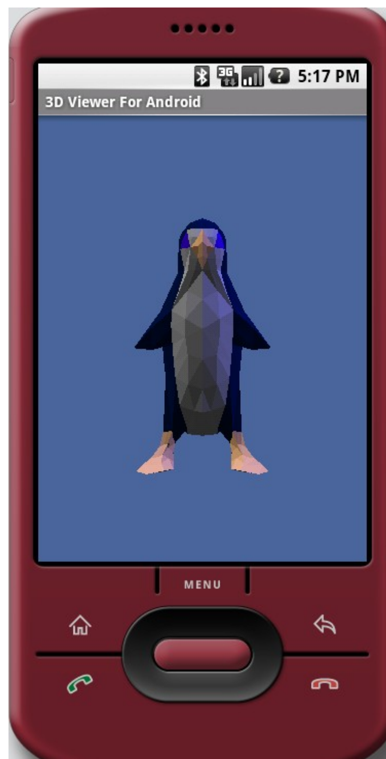

Figura 3.6 - Tela do 3DViewer

3.4 - Criação de Funções

O próximo passo após a finalização da primeira etapa era desenvolver as funções do 3DViewer. Foram estabelecidas três funções básicas que tornariam o programa um pouco mais interessante para o usuário. A primeira seria a possibilidade de escolha do arquivo DXF a ser renderizado. A segunda era poder rotacionar a imagem ao redor dos três eixos. Por fim, a última função seria permitir o uso de *zoom* na imagem.

A escolha do arquivo DXF foi implementada adicionando uma tela antes da tela principal do 3DViewer. Nessa nova tela seria possível escolher através do botão menu do celular qualquer um dos arquivos DXF contidos no diretório do 3DViewer.

Ao clicar no botão menu, uma busca por arquivos com extensão dxf é disparada dentro do diretório pertencente ao 3DViewer. Assim que a busca termina, uma lista das ocorrências achadas é mostrada ao usuário para que ele possa escolher um arquivo DXF para ser renderizado como pode ser visto na figura 3.7.

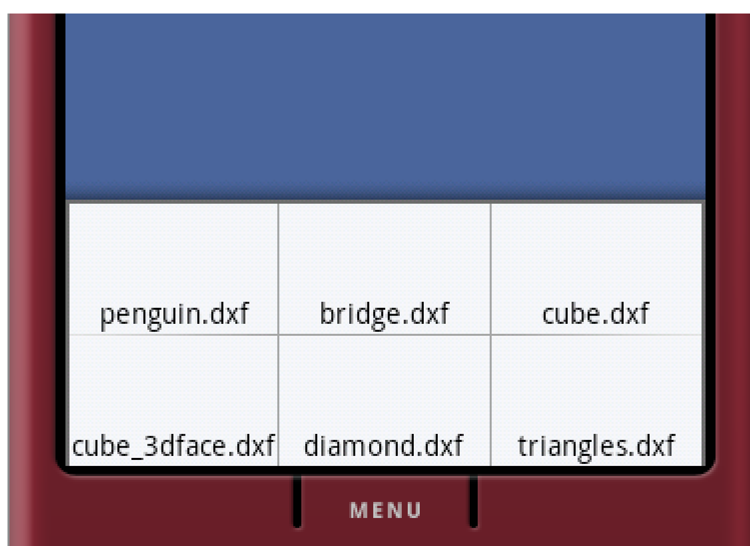

Figura 3.7 - Menu do 3DViewer para a escolha do arquivo DXF

A rotação da imagem ao redor dos eixos foi feita de duas formas. Uma delas é tocando e arrastando o dedo sobre a tela do celular na direção do eixo de rotação desejado e a outra é utilizando os botões do aparelho.

Como durante todo o desenvolvimento apenas o emulador de um celular foi usado, o movimento que simula o *touch screen* é feito com o *mouse*. Além disso, como

a tela é uma superfície plana, a imagem só poderia ser rotacionada ao redor dos eixos x e y ao utilizar esse método.

A rotação no OpenGL é feita através do método *glRotatef* que gira um objeto ao redor do vetor (x,y,z) conforme mostra o trecho de código abaixo.

```
(...)
gl.glRotatef(mRotateAngleX, 1.0f, 0.0f, 0.0f);
gl.glRotatef(mRotateAngleY, 0.0f, 1.0f, 0.0f);
gl.glRotatef(mRotateAngleZ, 0.0f, 0.0f, 1.0f);
(...)
```

Os atributos usados no primeiro argumento do método definem o ângulo em graus. Os três parâmetros restantes definem as coordenadas x, y e z do vetor unitário de rotação. Para que fosse possível rodar em cada direção com um ângulo diferente, o método foi utilizado três vezes seguidas passando um vetor em cada direção.

A definição dos ângulos de giro ao redor do eixo x e y é feita ao tocar o dedo na tela do celular. Nesse momento, o Android chama o método *onTouchEvent* que pode ser visto a seguir.

```
public boolean onTouchEvent(MotionEvent event) {
	switch (event.getAction()) {
		case MotionEvent.ACTION_DOWN:
			mOldX = event.getX();
			mOldY = event.getY();
			break;

		case MotionEvent.ACTION_MOVE:
			mActualX = event.getX();
			mActualY = event.getY();

			// Calculate delta and normalize
			mDeltaX = (mActualX - mOldX)/getWidth();
			mDeltaY = (mActualY - mOldY)/getHeight();
			mRotateAngleX += mDeltaY*mMaxRotateAngleX;
			mRotateAngleY += mDeltaX*mMaxRotateAngleY;

			mOldX = mActualX;
			mOldY = mActualY;
			break;
	}
	return true;
}
```

A posição é guardada ao tocar na tela e ao deslizar o dedo sobre ela. Sabendo das posições inicial e final do movimento é possível calcular a distância percorrida e consequentemente o ângulo de rotação proporcional à essa distância que se deve aplicar à imagem. Os atributos *mMaxRotateAngleX* e *mMaxRotateAngleY* definem os ângulos máximos de rotação que se pode fazer sem retirar o dedo da tela ao redor do eixo x e y, respectivamente.

Tanto a rotação ao redor do eixo z quanto o *zoom* só podem ser utilizados através do teclado do celular. Ao apertar os botões direcionais do celular, é possível alternar entre as funções existentes e alterar o valor de cada atributo relacionado à função. Assim, as teclas direita e esquerda escolhem umas das quatro possibilidades: rotação ao redor do eixo x, y, z ou *zoom*. Os botões sobe e desce, respectivamente, incrementam e decrementam o ângulo de rotação selecionado ou o *zoom*.

O *zoom* foi implementado utilizando o método *glTranslatef* cujo objetivo é mover todas as coordenas de um objeto ao longo dos eixos conforme pode ser visto abaixo.

```
(...)
gl.glTranslatef(0, 0, mZoom);
(...)
```

O código acima indica que não haverá translação nas direções x e y e que o objeto sofrerá uma translação na direção z que nesse caso é a profundidade. Por isso o incremento do atributo *mZoom* passa a impressão de que o objeto está aumentando ou diminuindo. Porém, o que realmente acontece é a aproximação ou afastamento do objeto em relação ao observador.

Conforme dito no seção 2.2, é possível aplicar uma transformação no objeto ou na câmera. Tanto a rotação quanto a translação estão sendo feitas diretamente no objeto enquanto a câmera permanece estática.

3.5 - Resultados

A versão final do 3DViewer ainda sofreu algumas melhorias antes que fosse concluído. Primeiro, na tela onde é possível escolher o arquivo DXF a ser renderizado foram acrescentadas as instruções de uso do software, como mostra a figura 3.8. Depois era preciso resolver a questão da espera do usuário até que a imagem fosse renderizada. Para isso, uma tela com uma barra de progresso foi adicionada. A barra é do tipo

indeterminada, como mostra a figura 3.9, pois não mostra ao usuário uma estimativa de tempo de carregamento da imagem. Sua função é mostrar que o programa está executando normalmente evitando a sensação de que está congelado.

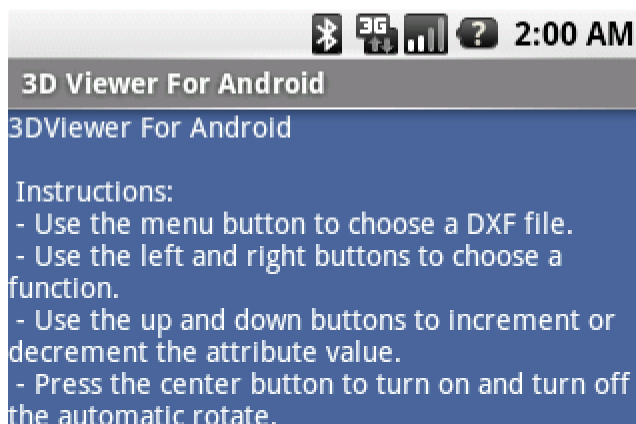

Figura 3.8 - Tela inicial do 3DViewer com as instruções de uso

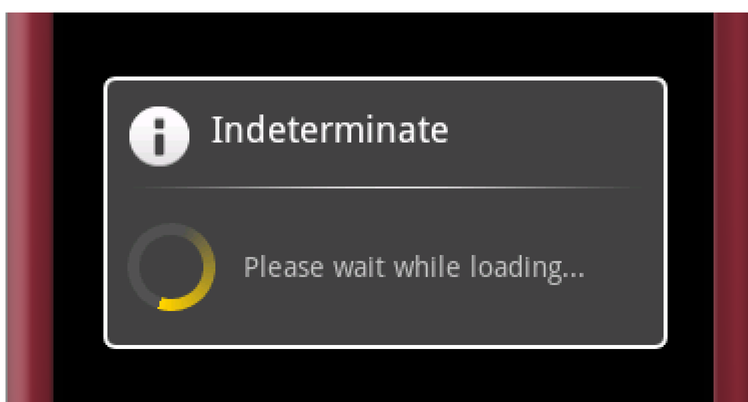

Figura 3.9 - Barra de progresso do 3DViewer

Por fim, algumas informações foram colocadas juntamente com a imagem assim que a mesma aparece. Logo acima do objeto, o usuário pode ver qual das quatro funções está selecionada no momento. Na parte inferior da tela é possível observar quanto vale cada ângulo de rotação, em graus, que está sendo aplicado em cada um dos eixos. A figura 3.10 mostra como ficou a tela principal do 3DViewer com um arquivo DXF carregado.

Figura 3.10 - Tela principal do 3DViewer com um arquivo DXF carregado

Capítulo 4
Conclusão

O desenvolvimento do 3DViewer possibilitou o estudo e a integração de tecnologias inovadoras como o Android e tecnologias consolidadas como o OpenGL. Essa já seria uma boa justificativa para querer trabalhar e estudar com o Android. Porém, o mundo vive a revolução dos dispositivos móveis e ainda não há uma grande empresa ou sistema que realmente domine esse mercado.

O Android foi desenvolvido pela Google, possui código aberto, oferece um *kit* de desenvolvimento para um dos mais populares ambientes de desenvolvimento e aceita aplicativos feitos em Java. Parece ser uma fórmula de sucesso infalível e que pode resultar no surgimento de um sistema para ser usado na maior parte dos dispositivos móveis do futuro.

No entanto, celulares com Android ainda são escassos ou simplesmente não existem em alguns países. Pode-se afirmar que é muito tempo para um sistema que foi lançado no fim de 2007.

A documentação ainda é pobre e confusa, principalmente na parte relacionada à OpenGL onde nem mesmo há referência para os métodos e atributos disponíveis no *framework*.

Foi possível constatar também a insatisfação da comunidade com relação à não disponibilização de um SDK para C++. É de se esperar que um sistema baseado em linux e com uma base de APIs em C++ possibilite o reaproveitamento de códigos feitos nessa linguagem.

Apesar de tudo isso, nada ainda está certo sobre como deve ser o sistema ideal para desenvolvimento de softwares para celulares. E o Android com certeza é um grande candidato.

Sobre trabalhos futuros envolvendo Android, OpenGL e até mesmo o 3DViewer, pode-se dizer que para desenvolver aplicativos utilizando OpenGL dentro do Android é necessário um código otimizado. Por exemplo, o 3DViewer utiliza triângulos para renderizar objetos na tela e cada triângulo é formado por três coordenadas. Mas é

possível desenhar dois triângulos com apenas quatro coordenadas. Sendo o segundo triângulo formado pela quarta coordenada juntamente com outras duas do primeiro triângulo.

O 3DViewer não usa esse algoritmo pelo fato do arquivo DXF conter as três coordenadas de cada vértice do triângulo. Porém, o programa poderia ser modificado deixando o carregamento e o redesenho da imagem mais rápida.

Poderiam também ser acrescentados outros parâmetros no programa deixando por conta do usuário a escolha dos seus valores. Além de permitir o download de um arquivo DXF ou de qualquer outro formato conhecido para que fosse renderizado.

O 3DViewer poderia ainda dar origem à um jogo 3D para celular ou à um software de simulação de ambientes utilizando os mesmos princípios vistos durante este projeto.

Bibliografia

[1] MANSSOUR, I. H., “Introdução à OpenGL – Profa. Isabel H. Manssour”, http://www.inf.pucrs.br/~manssour/OpenGL/Tutorial.html, 2005, (Acesso em 05 de abril de 2008)

[2] WOO, M.; NEIDER, J.; DAVIS, T.; SHREINER, D., OpenGL Programming Guide: the official guide to learning OpenGL, version 1.2. 3rd ed. Reading, Massachusetts: Addison Wesley, 1999. 730 p.

[3] KHRONOS GROUP, “OpenGL ES Reference Manual”, http://www.khronos.org/opengles/documentation/opengles1_0/html/index.html, 2003, (Acesso em maio de 2008)

[4] AUTODESK, “Autodesk - AutoCAD Services & Support - DXF Reference”, http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=12272454&linkID=10809853, 2008, (Acesso em maio de 2008)

[5] GOOGLE, “Android”, http://code.google.com/intl/pt-BR/android/, 2008, (Acesso em maio de 2008)

[6] GOOGLE, “Android Developers”, http://developer.android.com/index.html, 2009 (Acesso em janeiro de 2009)

[7] ZEUS COMMUNICATION, MULTIMEDIA & DEVELOPMENT, “Zeus CMD Design and Development Tutorials : OpenGL ES Programming Tutorials – Page 1”, http://www.zeuscmd.com/tutorials/opengles/index.php, 2005, (Acesso em abril de 2008)

[8] PINHO, M. S., “Biblioteca Gráfica OpenGL”, http://www.inf.pucrs.br/~pinho/CG/Aulas/OpenGL/OpenGL.html, (Acesso em março de 2008)

[9] CODE SOURCERY, “GNU Toolchain for ARM Processors”, http://www.codesourcery.com/sgpp/lite/arm, 2004, (Acesso em março de 2008)

[10] MODMYGPHONE, “Native C++ 'Hello World' working in emulator”, http://modmygphone.com/forums/showthread.php?t=90, 2007, (Acesso em março de 2008)