

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISUALIZADOR VOLUMÉTRICO DE IMAGENS DICOM NA
PLATAFORMA ANDROID

JONATAS DANIEL HERMANN

BLUMENAU
2013

2013/2-12

JONATAS DANIEL HERMANN

VISUALIZADOR VOLUMÉTRICO DE IMAGENS DICOM NA PLATAFORMA ANDROID

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

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

**BLUMENAU
2013**

2013/2-12

VISUALIZADOR VOLUMÉTRICO DE IMAGENS DICOM NA PLATAFORMA ANDROID

Por

JONATAS DANIEL HERMANN

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

Presidente:

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

Membro:

Prof. Mauro Marcelo Mattos, Dr. – FURB

Membro:

Prof. Aurélio Faustino Hoppe, M. Sc. – FURB

Blumenau, 10 de dezembro de 2013

Dedico este trabalho aos meus pais, que nunca mediram esforços para proporcionar uma educação de qualidade, sem a qual, a realização deste trabalho não seria possível.

AGRADECIMENTOS

À Deus, pelo dom da vida.

Aos meus pais, os principais responsáveis por eu ter chegado até aqui, por não terem medido esforços para que este trabalho e minha formação acontecessem. Vocês são a razão disso tudo.

À minha família, pela compreensão e apoio nas horas mais difíceis.

Aos guerreiros companheiros de curso, que sempre estiveram do meu lado, trocando ideias e experiências.

Aos meus amigos, que compreenderam a minha ausência e sempre se mostraram grandes apoiadores para que a conclusão deste trabalho se concretizasse.

Ao meu orientador, Dalton Solano dos Reis, pela paciência e por ter acreditado na conclusão deste trabalho.

Ao Marcelo da Mata Oliveira, pelo auxílio prestado.

A persistência é o menor caminho do êxito.

Charles Chaplin

RESUMO

Este trabalho apresenta a especificação e implementação de uma aplicação de visualização volumétrica de imagens DICOM para Android. O aplicativo lê os arquivos do exame médico computadorizado selecionado pelo usuário da aplicação e as imagens são extraídas de cada um destes arquivos. Cada imagem representa uma fatia. Estas fatias são apresentadas individualmente na visualização 2D em três direções distintas. Na visualização 3D elas são organizadas no espaço em três dimensões para formar o volume, que representa a parte do corpo da qual foi feito o exame.

Palavras-chave: Padrão DICOM. Visualização volumétrica. Android.

ABSTRACT

This document presents the specification and implementation of a volume rendering application of DICOM images for Android. The application reads the data of the scanned medical exam, selected by the user of the app, and the images are extracted from each of these data. Each image represents a slice. These slices are presented individually in the 2D view in three different directions. In the 3D view they are sorted on a space in three different dimensions in order to form the volume, which represents the part of the body that the exam was made on.

Key-words: DICOM Standard. Volume rendering. Android.

LISTA DE ILUSTRAÇÕES

Figura 1 - Exemplo de visualização volumétrica de um objeto	16
Figura 2 - Camadas do sistema operacional Android	18
Figura 3 - Visualização de uma imagem DICOM no aplicativo Osirix para iOS	20
Figura 4 - Diagrama de casos de uso	22
Quadro 1 - Caso de uso ler arquivos DICOM.....	22
Quadro 2 - Caso de uso visualizar volumetricamente	23
Quadro 3 - Caso de uso visualizar na direção sagital	23
Quadro 4 - Caso de uso visualizar na direção coronal.....	23
Quadro 5 - Caso de uso visualizar na direção axial	24
Figura 5 - Diagrama de classes do pacote activities	25
Figura 6 - Diagrama de classes do pacote adapters	26
Figura 7 - Diagrama de classes do pacote models	27
Figura 8 - Diagrama de classes do pacote reader.....	28
Figura 9 - Diagrama de classe do pacote view	29
Figura 10 - Diagrama de sequência	30
Quadro 6 - Trecho do código read da classe DicomImageReader.....	32
Quadro 7 - Método read16BitsImage da classe DicomImageReader	33
Quadro 8 - Método limiar da classe DicomImageReader.....	34
Quadro 9 - Método findMaxY da classe DicomImageReader.....	35
Quadro 10 – Método findMinY da classe DicomImageReader.....	35
Quadro 11 - Método findMaxX da classe DicomImageReader.....	35
Quadro 12 - Método findMinX da classe DicomImageReader	35
Quadro 13 - Trecho de código do método readImages da classe DicomReader	36
Quadro 14 - Método applyBoundingBox da classe DicomImage	37
Quadro 15 - Método setImage da classe CoronalContainer	38
Quadro 16 - Método setImage da classe AxialContainer	39
Quadro 17 - Método loadGLTextures da classe Square	41
Quadro 18 - Método draw da classe Square	41
Figura 11 - Tela de seleção de exames	43
Figura 12 - Tela de visualização em duas dimensões	44
Figura 13 - Alternando entre imagens na direção sagital	45

Figura 14 - Alternando entre imagens na direção coronal	46
Figura 15 - Alternando entre imagens na direção axial	47
Figura 16 - Tela de visualização volumétrica	48
Figura 17 - Ruído na imagem	50
Figura 18 - Gráfico do uso de memória utilizando o simulador	53
Figura 19 - Gráfico de uso de memória utilizando o dispositivo.....	54

LISTA DE TABELAS

Tabela 1 - Comparação entre este trabalho com o trabalho correlato.....	51
Tabela 2 - Consumo de memória da aplicação utilizando o simulador	53
Tabela 3 - Consumo de memória da aplicação utilizando o dispositivo.....	53
Tabela 4 - Desempenho por operação utilizando o simulador.....	55
Tabela 5 - Desempenho por operação utilizando o dispositivo	55

LISTA DE SIGLAS

DICOM - *Digital Imaging and COmmunications in Medicine*

UML - *Unified Modeling Language*

JAR - *Java Archive*

RGBA - *Red Green Blue Alpha*

JEE - *Java Enterprise Edition*

SDK - *Software Development Kit*

GC - *Garbage Collector*

SUMÁRIO

1 INTRODUÇÃO	13
1.1 OBJETIVOS DO TRABALHO.....	13
1.2 ESTRUTURA DO TRABALHO	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 VISUALIZAÇÃO VOLUMÉTRICA.....	15
2.2 PADRÃO DICOM.....	16
2.3 PLATAFORMA ANDROID	17
2.4 TRABALHOS CORRELATOS	18
2.4.1 Visualização Interativa 3D de Dados Volumétricos	18
2.4.2 Osirix	19
3 DESENVOLVIMENTO DA APLICAÇÃO	21
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO	21
3.2 ESPECIFICAÇÃO.....	21
3.2.1 Diagrama de casos de uso.....	21
3.2.1.1 Ler arquivos DICOM	22
3.2.1.2 Visualizar volumetricamente	23
3.2.1.3 Visualizar na direção sagital	23
3.2.1.4 Visualizar na direção coronal.....	23
3.2.1.5 Visualizar na direção axial	24
3.2.2 Diagrama de classes	24
3.2.2.1 Pacote activities	24
3.2.2.2 Pacote adapters	25
3.2.2.3 Pacote models.....	26
3.2.2.4 Pacote reader.....	27
3.2.2.5 Pacote view	28
3.2.3 Diagrama de sequência.....	29
3.3 IMPLEMENTAÇÃO.....	30
3.3.1 Técnicas e ferramentas utilizadas	30
3.3.2 Aplicação de visualização de imagens DICOM	31
3.3.2.1 Leitura dos arquivos DICOM.....	31
3.3.2.2 Visualização em duas dimensões	37

3.3.2.3 Visualização volumétrica	39
3.3.3 Operacionalidade da implementação.....	42
3.3.3.1 Visualização em duas dimensões	42
3.3.3.2 Visualização volumétrica	47
3.4 RESULTADOS E DISCUSSÃO.....	48
3.4.1 Testes da geração do volume.....	48
3.4.2 Comparação com o trabalho correlato.....	50
3.4.3 Memória e desempenho.....	51
3.4.3.1 Consumo de memória	52
3.4.3.2 Desempenho	54
4 CONCLUSÕES	56
4.1 EXTENSÕES.....	56
REFERÊNCIAS BIBLIOGRÁFICAS.....	57
APÊNDICE A – Relação dos formatos das apresentações dos trabalhos Error! Bookmark not defined.	
ANEXO A – Representação gráfica de contagem de citações de autores por semestre nos trabalhos de conclusões realizados no Curso de Ciência da Computação Error! Bookmark not defined.	

1 INTRODUÇÃO

Com o avanço da tecnologia na área medicinal, diversas tecnologias foram empregadas para um diagnóstico e estudo mais aprofundado do corpo humano. Este avanço permitiu que, hoje, tenham-se exames médicos de alta precisão com invasão praticamente nula ao paciente. Dentre esses exames, podem-se destacar a tomografia computadorizada e a ressonância magnética.

Tendo em vista este avanço, foi estabelecido com a finalidade de padronizar a formatação das imagens diagnósticas, um conjunto de normas para tratamento, armazenamento e transmissão de informação médica num formato eletrônico denominado *Digital Imaging and COmmunications in Medicine* (DICOM). O padrão, criado em 1983, possibilita o compartilhamento de imagens diagnósticas dos pacientes entre diversos laboratórios, mesmo tendo equipamentos de marcas distintas, desde que, façam uso do padrão especificado. Além do diagnóstico inicial é possível através do padrão DICOM acompanhar todo processo de evolução do paciente, já que ele contém informações detalhadas de seus diagnósticos (MONTEIRO, 2005, p. 54).

Aproveitando a popularidade e massificação dos dispositivos móveis, a área médica tem se aproveitado disso para auxiliar na visualização das imagens geradas por estes diagnósticos, para apresentar ao paciente, uma forma mais simples e clara das suas lesões.

Foi escolhida a plataforma Android devido, dentre outros fatores, ao seu recente crescimento e pela sua participação no mercado de dispositivos móveis, contendo, segundo a International Data Corporation (IDC) (2012), 68,1% da fatia deste mercado. Outro fator determinante na escolha da plataforma foi a percepção de certa deficiência nas ferramentas disponíveis para o tema proposto, tornando assim o trabalho mais desafiante e maior impacto na comunidade.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um aplicativo para a plataforma Android que permita realizar a visualização volumétrica de imagens no padrão DICOM.

Os objetivos específicos do trabalho são:

- a) utilizar as imagens DICOM armazenadas localmente para gerar as visualizações;
- b) inspecionar em 2D e visualizar em 3D as imagens volumétricas de diferentes pontos de vista.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos. O segundo capítulo aborda a fundamentação teórica necessária para a compreensão deste trabalho.

No terceiro capítulo são apresentadas as etapas de desenvolvimento da aplicação. Primeiramente são apresentados os requisitos da aplicação. Posteriormente é mostrada a especificação da aplicação com os diagramas desenvolvidos. No passo seguinte são descritas as ferramentas e técnicas utilizadas na implementação e também a operacionalidade da aplicação. Por fim são apresentados os resultados e discussão.

No quarto capítulo são apresentadas as conclusões e sugestões para trabalhos futuros que possam ser desenvolvidos a partir deste.

2 FUNDAMENTAÇÃO TEÓRICA

Inicialmente, na seção 2.1 são apresentados os conceitos de visualização volumétrica. Na seção 2.2, tem-se uma breve explanação sobre o padrão DICOM e porque do seu surgimento e importância na área médica nos dias de hoje. Posteriormente é apresentado a história e conceitos iniciais da plataforma Android, mostrando em que base tecnológica o sistema será desenvolvido, bem como a biblioteca principal no desenvolvimento do visualizador volumétrico. Finalizando, são apresentados dois trabalhos correlatos que servem como auxílio no desenvolvimento do trabalho aqui especificado.

2.1 VISUALIZAÇÃO VOLUMÉTRICA

Visualização é um termo relacionado aos métodos que permitem a extração de informações relevantes a partir de complexos conjuntos de dados, processo geralmente feito através da utilização de técnicas de computação gráfica e processamento de imagens (PAIVA; SEIXAS; GATTAS, 1999, p. 01).

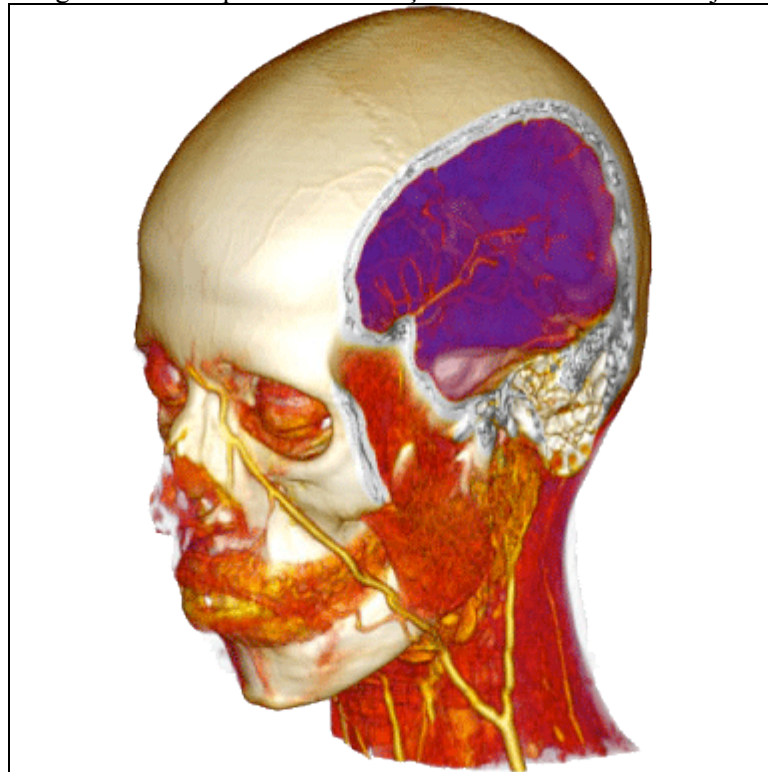
Denomina-se visualização científica quando estes conjuntos de dados representam fenômenos complexos e o objetivo é a extração de informações científicas relevantes. Uma das mais interessantes subáreas da visualização científica, e que tem tido um rápido crescimento, é a visualização volumétrica.

Visualização volumétrica é o conjunto de técnicas utilizadas na visualização de dados associados a regiões de um volume, tendo como principal objetivo a exibição do interior de objetos volumétricos, a fim de explorar sua estrutura e facilitar sua compreensão.

Os dados volumétricos podem ser capturados através da amostragem em três dimensões de um objeto real, usando diferentes dispositivos como: satélites, scanners, tomógrafos, simuladores, medidores especiais, entre outros. Estes dados são geralmente tratados como uma matriz de elementos de volume, denominados voxels. Voxels são paralelepípedos, fortemente agrupados, formados pela divisão do espaço do objeto através de um conjunto de planos paralelos aos eixos principais desse espaço (MCCORMICK; DEFANTI; BROWN, 1987).

Na Figura 1 tem-se um exemplo de um software realizando a visualização volumétrica de um objeto.

Figura 1 - Exemplo de visualização volumétrica de um objeto



Fonte: Vienna University of Technology (2013).

2.2 PADRÃO DICOM

O padrão DICOM surgiu em 1983, por um comitê criado pela *American College of Radiology* (ACR) e pela *National Electrical Manufacturers Association* (NEMA), visando padronizar a comunicação, apresentação e armazenamento das imagens médicas (MONTEIRO, 2005, p. 54).

O padrão estabelece uma linguagem comum entre todos os equipamentos da área médica, mesmo sendo os equipamentos, de marcas diferentes. Esta linguagem contém uma série de regras que permitem que as imagens sejam trocadas por diversos aparelhos de imagens diagnósticas entre diferentes hospitais, laboratórios, clínicas, entre outros.

Ele diferencia-se dos outros formatos de imagens por permitir que as informações dos pacientes sejam armazenadas juntamente com a imagem, mas de forma estruturada. Isso é, elas são armazenadas contendo identificadores, conhecidos como “tags”, que identificam e limitam as informações. A imagem propriamente dita, no padrão DICOM é baseada no formato JPEG, com ou sem compressão, dependendo do equipamento que a gerou (MONTEIRO, 2005, p. 54).

Atualmente, o padrão DICOM encontra-se na versão 3.0, que foi apresentada em 1993, e desde então, tem se tornado padrão em muitos equipamentos de imagens diagnósticas (MONTEIRO, 2005, p. 54).

2.3 PLATAFORMA ANDROID

Android é um sistema operacional móvel baseado no Linux desenvolvido pela Google, tendo sua primeira versão distribuída em 2007.

A plataforma Android foi construída de forma que permite que os desenvolvedores possam criar aplicações capazes de tirar o melhor proveito de tudo que um dispositivo móvel pode oferecer. Não existe diferenciação para o Android entre uma aplicação nativa e uma aplicação criada por terceiros, todas tem acesso aos mesmos recursos do dispositivo. Devido a isso, cada usuário pode customizar seu dispositivo da forma que melhor lhe agradar, trocando, por exemplo, a aplicação de discador do telefone, o seu navegador de Internet ou até mesmo a interface principal do sistema. Conforme Lecheta (2010, p. 22), esta característica, associada ao fato do Android ser uma plataforma de código aberto, atraiu o interesse de diversas empresas, que viram no Android uma oportunidade de terem um sistema customizado em seus dispositivos.

Com isso surgiu a *Open Handset Alliance* (OHA), um grupo formado por diversas empresas na área de tecnologia, que ajudam a manter e distribuir o sistema operacional, tendo como principal desenvolvedora, a própria Google.

Segundo a Google (2013), o sistema operacional é dividido em quatro principais camadas: *kernel* Linux, bibliotecas, *framework* de aplicação e aplicações. Na Figura 2 pode-se ver como essas camadas dividem-se.

Figura 2 - Camadas do sistema operacional Android



Fonte: Android Developers (2009).

Como pode ser visto na Figura 2, a camada de bibliotecas já fornece suporte à biblioteca *OpenGL for Embedded Systems* (OpenGL-ES), que pode ser usado na concepção da imagem a partir do arquivo DICOM escolhido. A biblioteca OpenGL-ES embutida no Android encontra-se atualmente na versão 3.0.2, baseada na versão 4 do OpenGL. Alguns recursos da versão OpenGL foram retirados na versão OpenGL-ES, visando otimização para sistemas embarcados.

2.4 TRABALHOS CORRELATOS

Existem alguns trabalhos acadêmicos relacionados ao tema de visualização volumétrica que podem servir como fonte de pesquisa. Dentre eles foram selecionados os trabalhos “Visualização Interativa 3D de Dados Volumétricos” (CARNEIRO; MARTHA, 2000) e o software Osirix (2013).

2.4.1 Visualização Interativa 3D de Dados Volumétricos

O trabalho apresenta, inicialmente uma visão geral de *rendering* de volumes, mostrando aspectos importantes como pipeline de visualização volumétrica direta. A seguir é estudado os mecanismos mais usuais de interação 3D, além de apresentar alguns recursos que

facilitam a interação, tais como alinhadores de campo de gravidade. Posteriormente é feito um estudo mais detalhado sobre algumas ferramentas de interação 3D, em especial o *Open Inventor*, *Virtual Reality Modelling Language* (VRML), *Widgets 3D* e o MTK (CARNEIRO; MARTHA, 2000, p. 22).

Por fim, o trabalho apresenta um estudo da interação 3D em ambientes volumétricos, sempre procurando identificar as técnicas de interação que possam permitir a visualização exploratória dos dados, já que esse é um dos principais objetivos dos sistemas de visualização volumétrica, principalmente os voltados para a área médica e engenharia (CARNEIRO; MARTHA, 2000, p. 33).

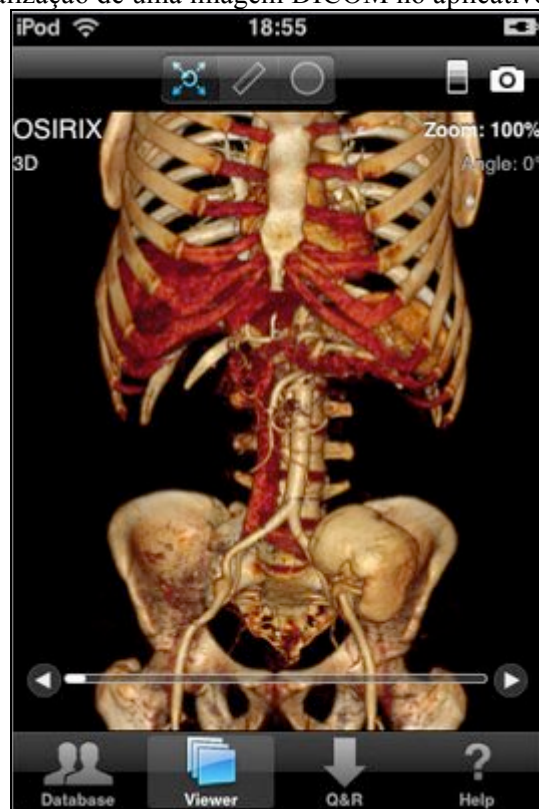
Apesar de todo estudo, a conclusão chegada foi que não há muitas novidades na área de visualização volumétrica, porém há muito campo à ser explorado na área de realidade virtual.

2.4.2 Osirix

O Osirix é conhecido como a mais completa ferramenta para visualização de imagens DICOM. Sendo desenvolvido na linguagem Objective-C, têm o seu código como *open-source*. Esta ferramenta foi desenvolvida para a plataforma Mac-OS, tendo sua distribuição gratuita. Posteriormente foi lançada uma versão para iOS, sendo essa paga. Dentre as principais funcionalidades desta ferramenta estão a possibilidade de visualizar todos os metadados que constam no arquivo DICOM e suas diversas modalidades de visualização das imagens DICOM, sendo possível realizar a visualização em 2D, 3D e para alguns tipos de imagens médicas é possível ainda obter uma visualização 4D e 5D. Esta ferramenta também permite visualizar as imagens em orientações diferentes e possui uma funcionalidade que permite realizar a remoção de ossos quando é realizada uma visualização volumétrica da imagem (OSIRIX, 2013).

Na Figura 3 é possível verificar uma das visualizações na plataforma iOS.

Figura 3 - Visualização de uma imagem DICOM no aplicativo Osirix para iOS



Fonte: Apple iTunes (2012).

3 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo são expostas as etapas do desenvolvimento da aplicação de visualização de imagens DICOM e da biblioteca para realizar a leitura do arquivo DICOM assim como suas imagens. São apresentados os principais requisitos, a especificação, a implementação e ao fim os resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O aplicativo para a visualização volumétrica de imagens DICOM possui os seguintes requisitos:



- a) efetuar a leitura do cabeçalho e das imagens de um arquivo DICOM (Requisito Funcional – RF);
- b) apresentar a sequencia de imagens no formato 2D contidas no arquivo DICOM (RF);
- c) efetuar a renderização volumétrica das imagens DICOM no próprio dispositivo (RF);
- d) ser implementado utilizando a linguagem de programação Java (Requisito Não-Funcional – RNF);
- e) ser implementado utilizando o ambiente de desenvolvimento Eclipse (RNF);
- f) ser desenvolvido para executar em dispositivos móveis com o sistema operacional Android (RNF).

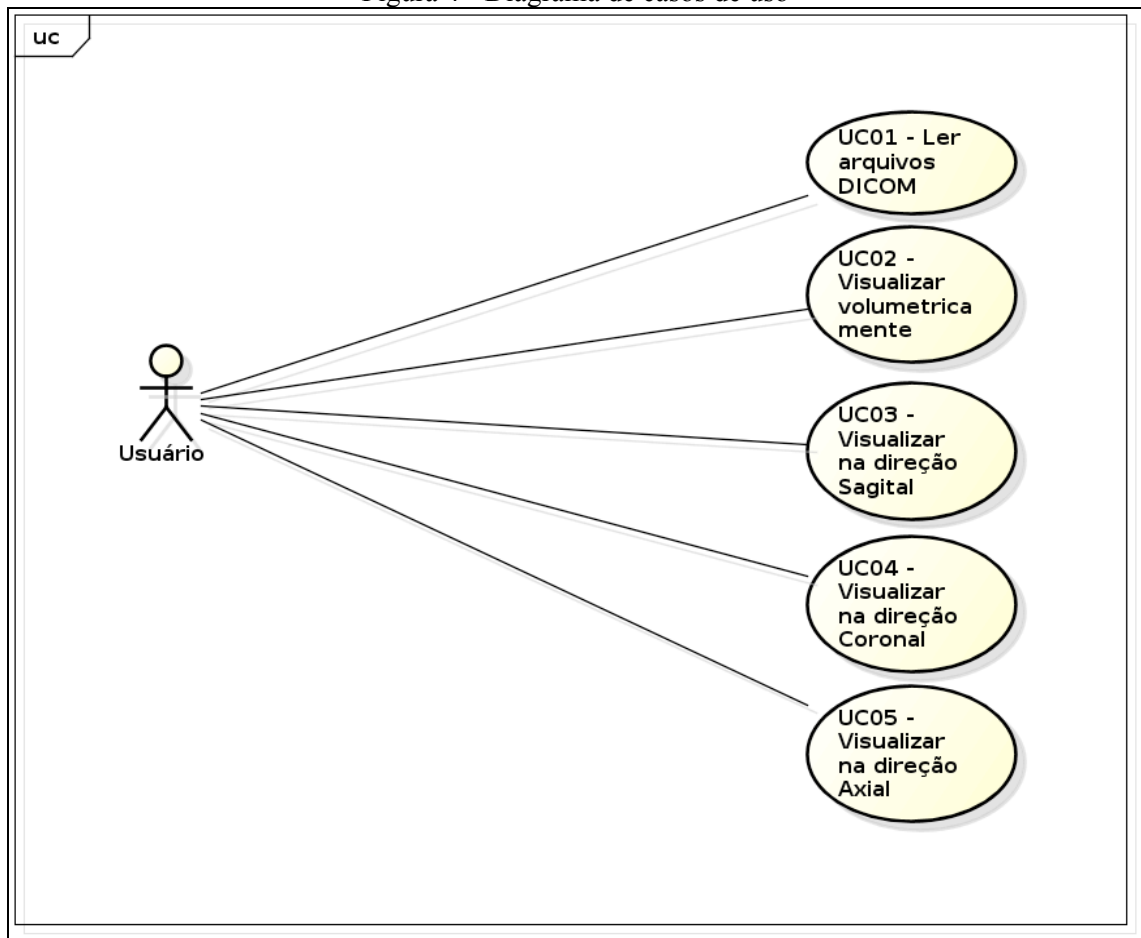
3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi feita utilizando modelagem de diagrama de casos de uso, diagrama de classes e diagrama de sequência, todos da *Unified Modeling Language* (UML) utilizando a ferramenta Astah. A seguir são apresentados os diagramas.

3.2.1 Diagrama de casos de uso

Nesta seção são descritos os casos de uso de todas as funcionalidade da aplicação. Foi identificado somente um ator que interage com a aplicação. Ele foi identificado como Usuário, que utilizará todas as funcionalidades da aplicação. Na Figura 4 pode-se observar o diagrama com o ator e os casos de uso.

Figura 4 - Diagrama de casos de uso



3.2.1.1 Ler arquivos DICOM

Este caso de uso descreve qual é a relação entre o usuário e a funcionalidade que possibilita ler arquivos DICOM. O Quadro 1 descreve os detalhes deste caso de uso.

Quadro 1 - Caso de uso ler arquivos DICOM

Ler arquivos DICOM	
Pré-condições	Estar com a tela de seleção de exame aberta.
Cenário	<ol style="list-style-type: none"> 1) O usuário seleciona um exame. 2) A aplicação obtém o diretório do exame. 3) A aplicação obtém todos os arquivos DICOM do diretório. 4) A aplicação lê todos os arquivos. 5) A aplicação obtém as imagens de todos os arquivos.
Pós-condições	A tela de seleção de exame é fechada.



3.2.1.2 Visualizar volumetricamente

Este caso de uso descreve qual é a relação entre o usuário e a visualização volumétrica de imagens DICOM. O Quadro 2 descreve em detalhes este caso de uso.

Quadro 2 - Caso de uso visualizar volumetricamente

Visualizar volumetricamente	
Pré-condições	O usuário deve estar com a tela de visualização em duas dimensões aberta.
Cenário	1) A aplicação visualiza volumetricamente o exame.
Pós-condições	A tela de visualização em duas dimensões é aberta.

3.2.1.3 Visualizar na direção sagital

Este caso de uso descreve qual é a relação entre o usuário e a visualização em duas dimensões de imagens DICOM na direção sagital. O Quadro 3 descreve em detalhes este caso de uso.

Quadro 3 - Caso de uso visualizar na direção sagital

Visualizar na direção sagital	
Pré-condições	O usuário deve ter selecionado um exame na tela de exames.
Cenário	1) A aplicação exibe a imagem do exame na direção sagital. 2) O usuário altera o componente <code>slider</code> do quadro sagital. 3) A aplicação exibe a imagem correspondente à posição atual do componente <code>slider</code> .
Pós-condições	A tela de visualização volumétrica é aberta.

3.2.1.4 Visualizar na direção coronal

Este caso de uso descreve qual é a relação entre o usuário e a visualização em duas dimensões de imagens DICOM na direção coronal. O Quadro 4 descreve em detalhes este caso de uso.

Quadro 4 - Caso de uso visualizar na direção coronal

Visualizar na direção coronal	
Pré-condições	O usuário deve ter selecionado um exame na tela de exames.
Cenário	1) A aplicação exibe a imagem do exame na direção coronal.

	2) O usuário altera o componente <code>slider</code> do quadro coronal. 3) A aplicação exibe a imagem correspondente à posição atual do componente <code>slider</code> .
Pós-condições	A tela de visualização volumétrica é aberta.

3.2.1.5 Visualizar na direção axial

Este caso de uso descreve qual é a relação entre o usuário e a visualização em duas dimensões de imagens DICOM na direção axial. O Quadro 5 descreve em detalhes este caso de uso.

Quadro 5 - Caso de uso visualizar na direção axial

Visualizar na direção axial	
Pré-condições	O usuário deve ter selecionado um exame na tela de exames.
Cenário	1) A aplicação exibe a imagem do exame na direção axial. 2) O usuário altera o componente <code>slider</code> do quadro axial. 3) A aplicação exibe a imagem correspondente à posição atual do componente <code>slider</code> .
Pós-condições	A tela de visualização volumétrica é aberta.

3.2.2 Diagrama de classes

Nesta seção é apresentada a especificação das classes que constituem a aplicação de visualização de imagens DICOM. As classes foram separadas em cinco pacotes, divididos em `activities`, `adapters`, `models`, `reader` e `view`. Descreve-se cada um deles em seguida.

3.2.2.1 Pacote `activities`

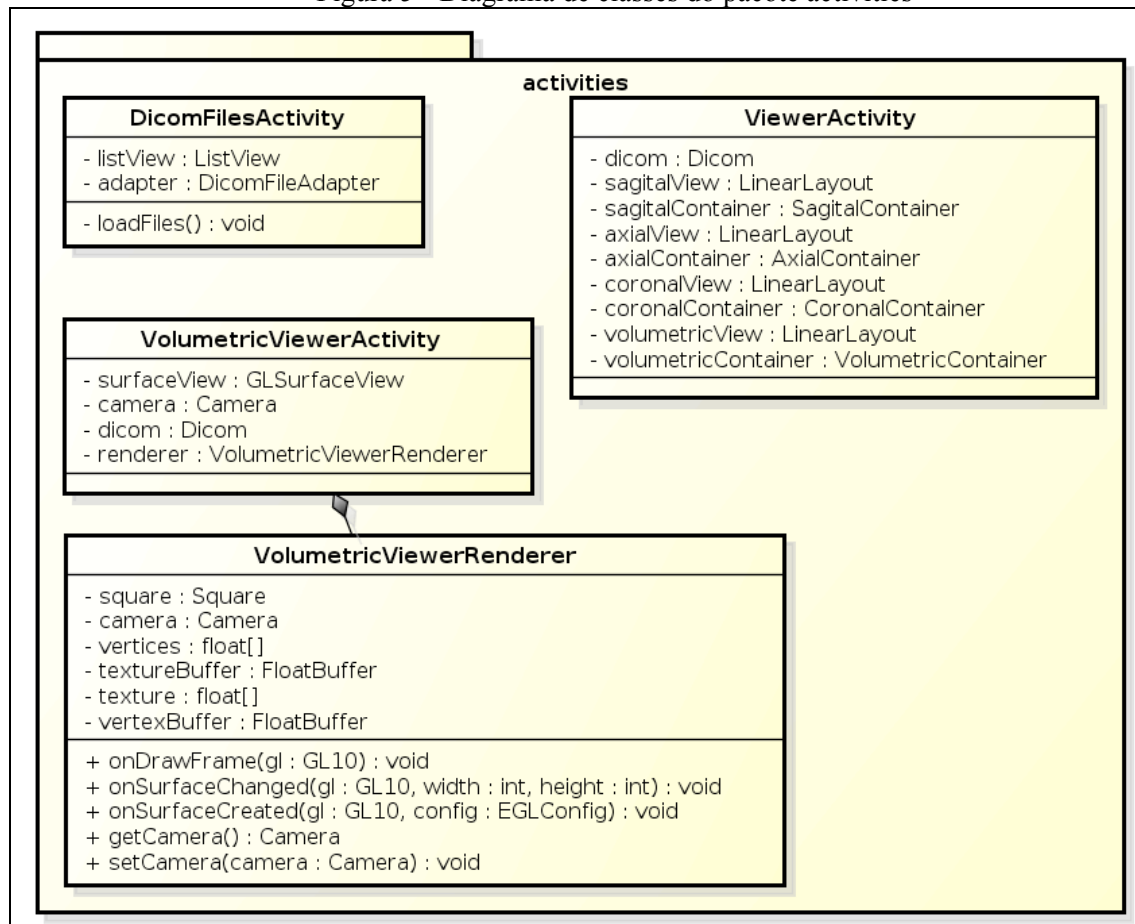
Neste pacote encontram as classes responsáveis por exibir o conteúdo visual para o usuário. Cada `activity` representa uma tela do aplicativo.

No início da execução do aplicativo, é exibido a `activity DicomFilesActivity`, que em uma lista, apresenta os exames disponíveis para visualização, carregados através do método privado `loadFiles`.

Selecionado um exame, a `activity ViewerActivity` é chamada, recebendo como parâmetro o diretório do exame selecionado. Essa classe é responsável por chamar as classes de leitura e por exibir a tela com visualização em duas dimensões nas direções sagital, axial e coronal. A partir dessa classe pode-se também chamar a tela de visualização volumétrica.

A classe responsável por visualizar o objeto volumetricamente é a `VolumetricViewerActivity`. Essa classe utiliza um component especial para o uso do OpenGL-ES, chamado `GLSurfaceView`, que requer uma instância de uma classe filha da interface do Android, chamada `GLSurfaceView.Renderer` que cuida do ciclo de execução do OpenGL-ES. Sendo assim, a classe `VolumetricViewerRenderer` foi criada, implementando a interface `GLSurfaceView.Renderer` e possui os métodos herdados `onDrawFrame`, `onSurfaceCreated` e `onSurfaceChanged`.

Figura 5 - Diagrama de classes do pacote activities

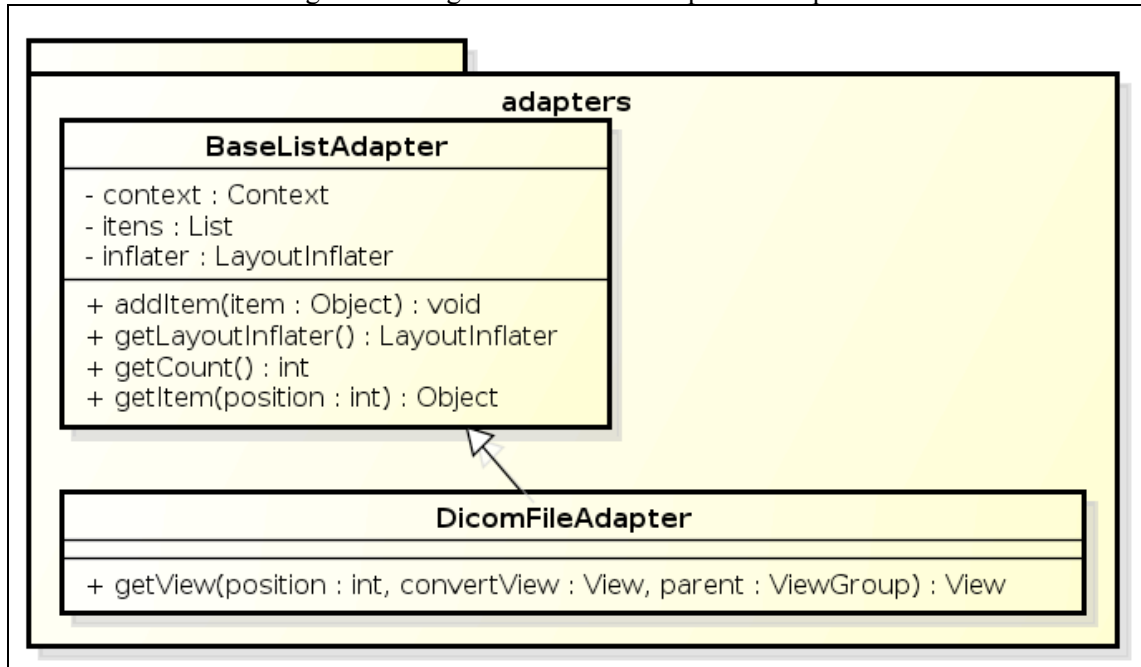


3.2.2.2 Pacote adapters

O Android usa um conceito de `Adapters` para interligar um componente de tela com o dado propriamente dito. No caso do aplicativo especificado, teremos um *adapter* para intermediar a nossa lista de exames e o dado que cada item da lista terá. Pensando em extensões do aplicativo, o pacote `adapters` possui uma classe chamada `BaseListAdapter` que agrupa alguns comportamentos que serão padrão para a maioria dos *adapters* criados. É o

caso da classe `DicomFileAdapter` que exibirá os dados necessários para a seleção do exame na tela inicial do aplicativo.

Figura 6 - Diagrama de classes do pacote adapters



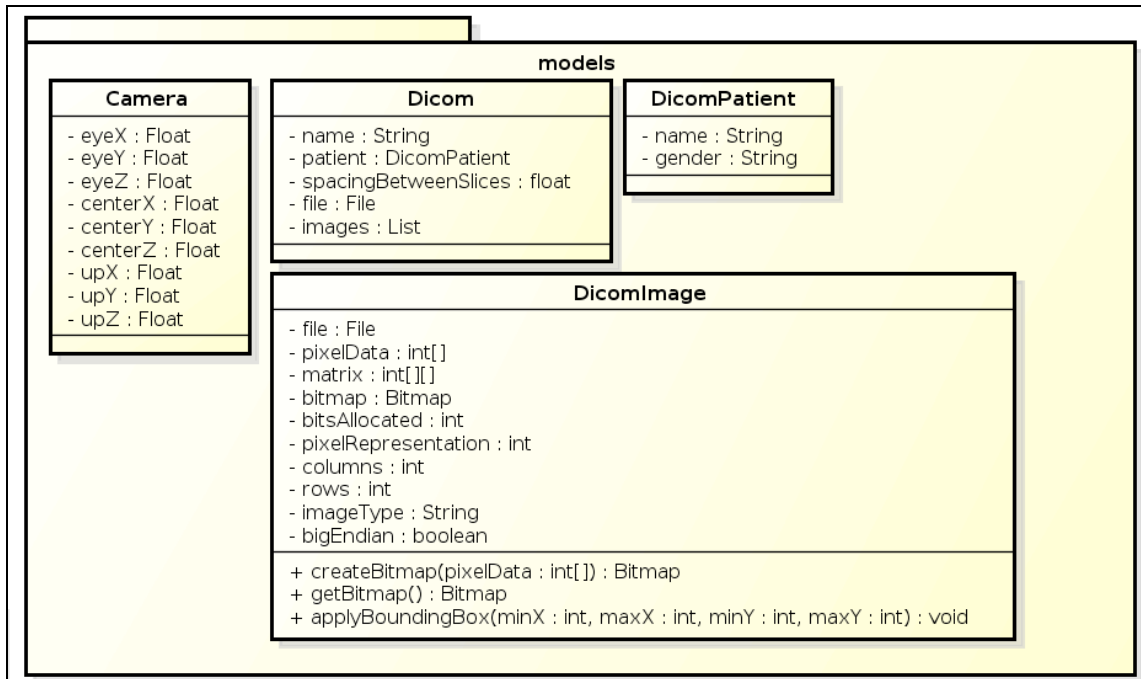
3.2.2.3 Pacote `models`

Este pacote contém as classes que representam os dados obtidos do exame assim como uma representação da câmera para a sua movimentação em relação ao objeto em cena.

A classe `Dicom` contém os dados principais do exame, assim como uma lista com a representação das imagens obtidas do exame, representadas pela classe `DicomImage`. Além disso, ainda é possível através da classe `Dicom`, obter uma instância da classe `DicomPatient` que contém dados básicos do paciente.

A classe `Camera` foi criada para agrupar os atributos da câmera, necessários para o funcionamento do posicionamento da câmera em relação ao objeto.

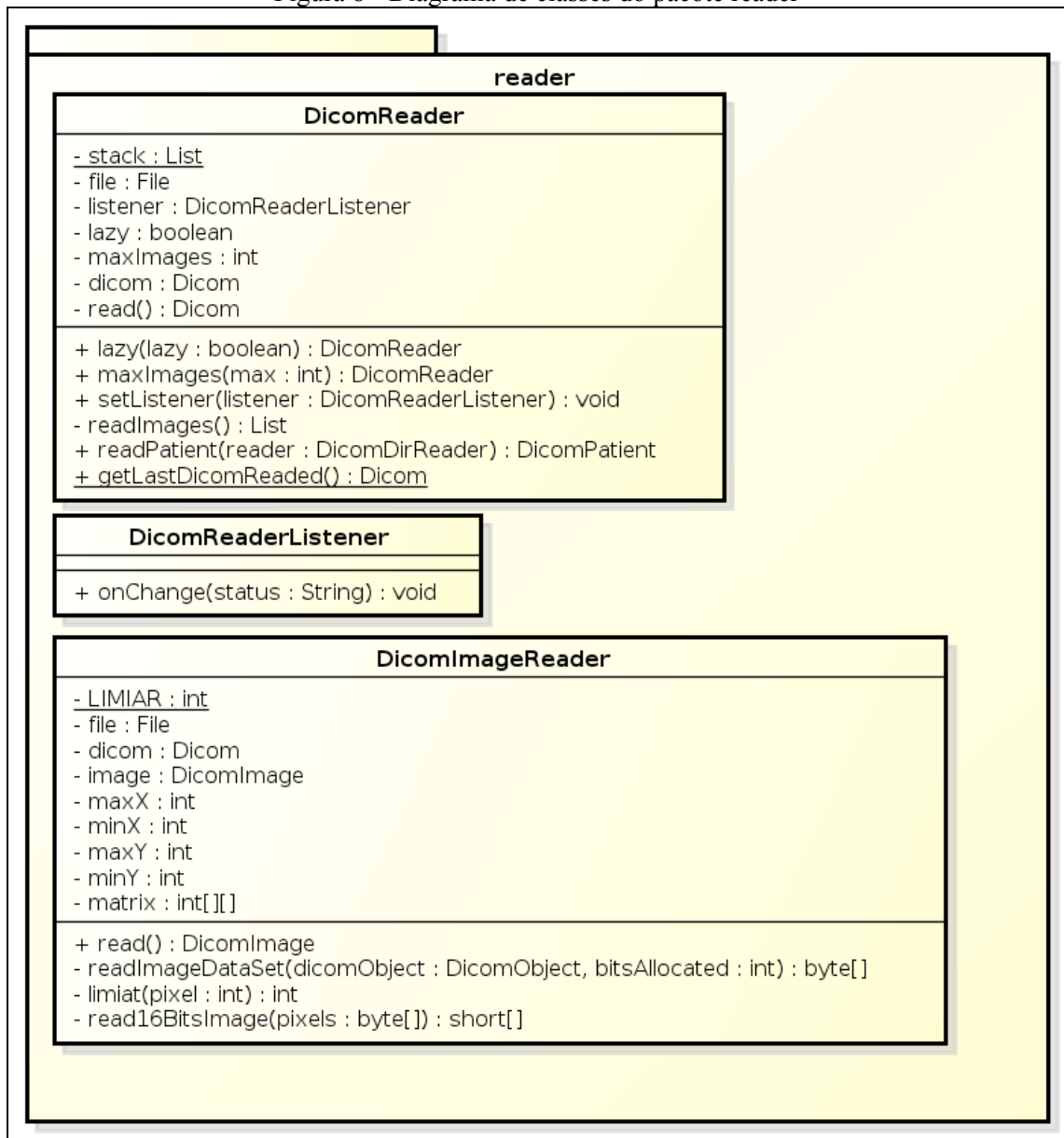
Figura 7 - Diagrama de classes do pacote models



3.2.2.4 Pacote reader

O pacote `reader` contém apenas três classes, responsáveis por fazer a leitura dos dados do exame assim como suas imagens. Os algoritmos de conversão e aplicação da `bounding-box` nas imagens também estão contidos na classe `DicomImageReader` para evitar processamento e uso de memória extra. A interface `DicomReaderListener` é utilizada para reportar a classe chamadora o *status* da leitura do arquivo. Abaixo tem-se a representação do diagrama de classes do pacote.

Figura 8 - Diagrama de classes do pacote reader



3.2.2.5 Pacote view

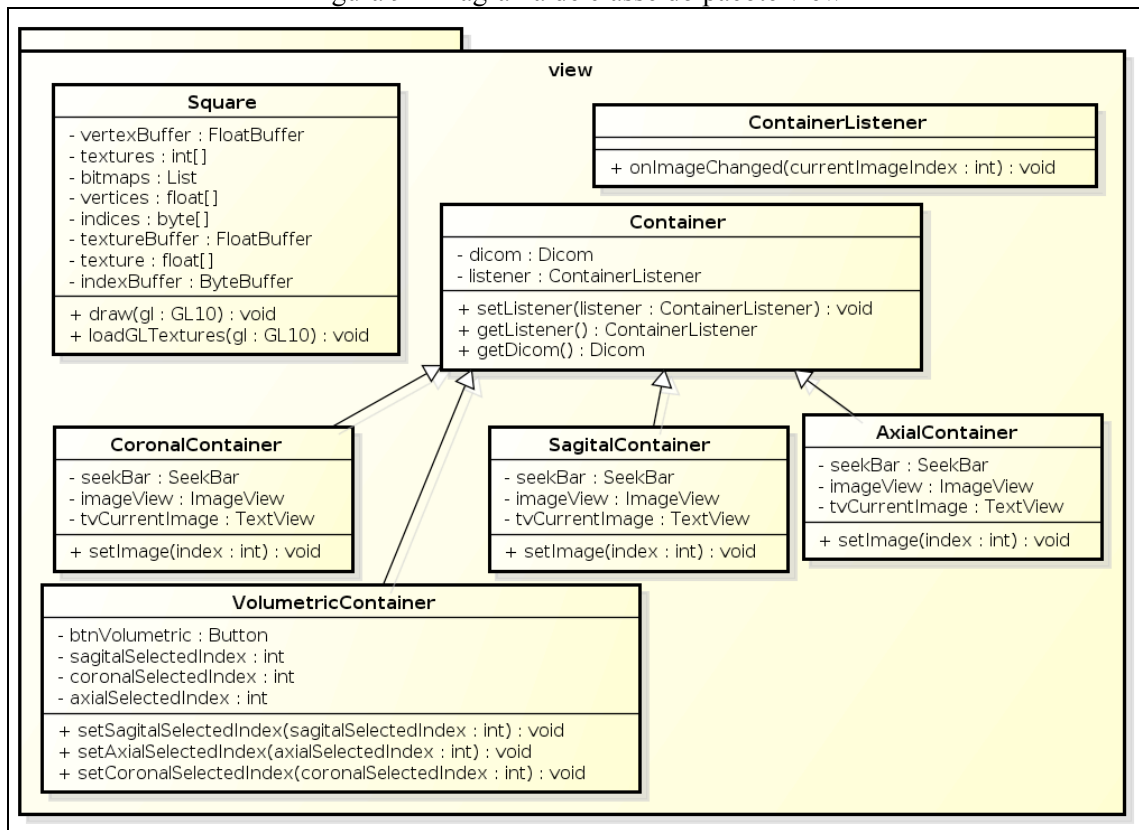
O pacote `view` contém as classes responsáveis por exibir o conteúdo visual. Para o uso na tela de visualização em duas dimensões, criou-se uma classe base chamada `Container`, onde tem-se agrupado os comportamentos e valores padrões para cada um dos quatro espaços que são exibidos na tela. Para cada um dos espaços utilizados, há uma classe específica, que herda de `Container`. Entre elas, as classes `SagitalContainer`, `CoronalContainer` e `AxialContainer` tem a função de exibir as imagens em duas dimensões, cada uma na direção sagital, coronal e axial respectivamente. A estrutura das três classes é idêntica, diferenciando-se apenas pelo algoritmo usado na visualização das imagens

na direção especificada. Ainda herdando de `Container`, tem-se a classe `VolumetricContainer`, que contém apenas um botão que faz a chamada para a tela de visualização volumétrica.

Finalmente temos a classe chamada `Square`, que será responsável por desenhar as superfícies quadradas na tela, onde serão aplicados as texturas de cada imagem obtida do exame.

O método `loadGLTextures` é o responsável por carregar inicialmente todas as texturas das imagens para dentro do objeto, e posteriormente, o método `draw` é chamado pelo método `onDrawFrame` da classe `ViewerRenderer`, que é responsável por desenhar as superfícies quadradas na cena e aplicar a textura carregada em cada um deles.

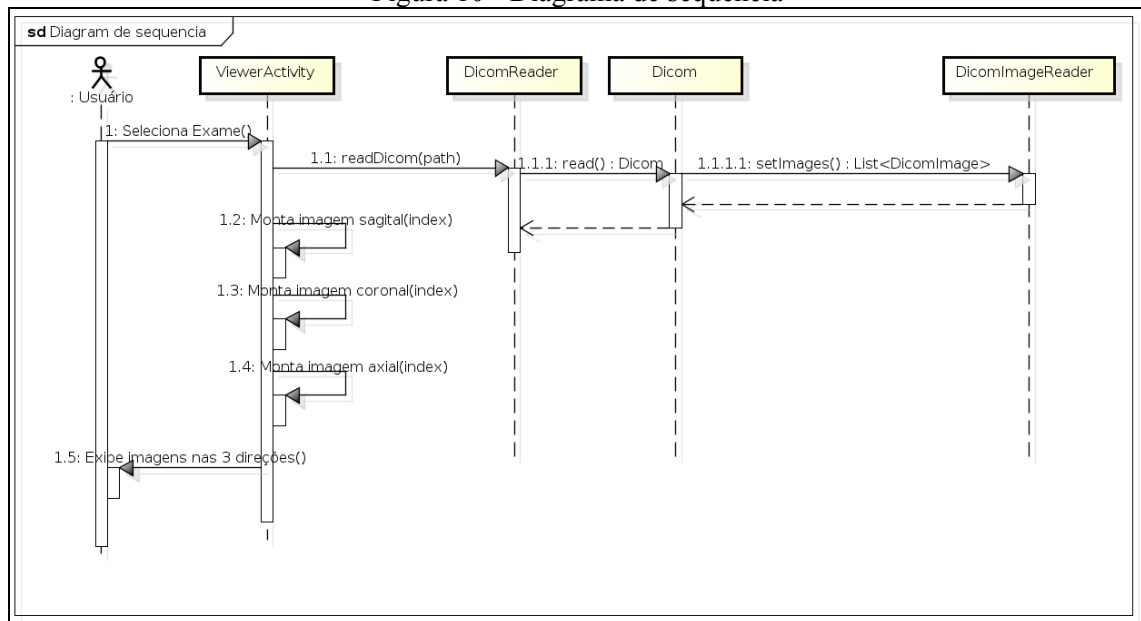
Figura 9 - Diagrama de classe do pacote view



3.2.3 Diagrama de sequência

Esta seção apresenta o diagrama de sequência, demonstrando como ocorre o processo de leitura dos arquivos DICOM até a exibição das imagens em duas dimensões, demonstrado em maiores detalhes na Figura 10.

Figura 10 - Diagrama de sequência



Nesta interação, o usuário seleciona um exame na tela de seleção de exames, e a classe `ViewerActivity` é chamada. Esta irá chamar o método `read` da classe `DicomReader`, através do método `readDicom`, passando como parâmetro, o diretório do exame selecionado. Por sua vez, a classe `DicomReader` chamará através do método `setImages`, a classe `DicomImageReader`, que ficará responsável por fazer a leitura das imagens. Terminado o processo de leitura, a classe `DicomReader` irá retornar um objeto do tipo `Dicom` a classe `ViewerActivity`, que de posse desse objeto, irá montar as superfícies quadradas na tela com as imagens na direção sagital, coronal e axial.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas na implementação. Também serão descritos detalhes das classes e algoritmos implementados. Por fim, a operacionalidade da implementação é apresentada.

3.3.1 Técnicas e ferramentas utilizadas

A linguagem utilizada no desenvolvimento do aplicativo foi Java e o ambiente utilizado foi o Eclipse. A única biblioteca de terceiros utilizado, foi a DCM4CHE, utilizada para ler as informações do arquivo DICOM assim como suas imagens. Foi utilizada também a biblioteca OpenGL-ES, que vem por padrão com as bibliotecas do Android.

Para criação da interface do usuário foi utilizada as classes provenientes do próprio *framework* de desenvolvimento para Android.

Como não havia uma versão específica para Android da biblioteca DCM4CHE, foi extraído alguns *Java Archive's* (JAR) para o correto funcionamento num ambiente mobile como o Android. Além disso, nenhuma outra adaptação foi necessária.

3.3.2 Aplicação de visualização de imagens DICOM

Primeiramente, são lidos os diretórios que contém os arquivos necessários para a leitura do exame. No momento, esses diretórios estão fixos no código. A partir da seleção de um desses exames, o aplicativo faz a leitura de todas as informações necessárias para a apresentação, utilizando as classes `DicomReader` e `DicomImageReader`, populando com as informações obtidas, as classes `Dicom` e `DicomImage`.

Para a visualização em duas dimensões, nas direções sagital, coronal e axial, a classe `ViewerActivity` é chamada, apresentando os resultados de cada imagem obtida.

A visualização volumétrica do objeto fica a cargo da classe `VolumetricViewerActivity`, que possui uma instância da classe `VolumetricViewerRenderer`, que é necessária para o funcionamento correto do OpenGL-ES.

3.3.2.1 Leitura dos arquivos DICOM

Nesta etapa, os diretórios pré-definidos pela aplicação são obtidos, e a partir deles, todas as informações necessárias do exame bem como suas imagens, são capturadas.

Para a leitura das informações do arquivo DICOM, foi utilizada partes da biblioteca DCM4CHE.

No diretório do exame, a classe `DicomReader`, por meio do método `read`, irá procurar um arquivo `DICOMDIR`, para a partir desse arquivo, realizar a leitura dos dados do paciente e das imagens do exame.

Para a leitura das imagens, notou-se na maioria dos exames testado, a presença do diretório `DICOM`, onde estava contido os arquivos de imagens do exame. Com essa constatação, as imagens lidas, são as presentes dentro do diretório `DICOM`, que por sua vez, está contido no mesmo diretório onde se encontra o arquivo `DICOMDIR`.

Para a leitura das imagens, foi obtido um `array` de `bytes`, através da constante `Tag.PixelData` da biblioteca DCM4CHE. Além dos `bytes` obtidos, outras informações são obtidas e armazenadas no objeto da classe `DicomImage`, tais como os `bits` alocados para a imagem e o número de colunas e linhas da imagem. Este método é apresentado no Quadro 6.

Quadro 6 - Trecho do código read da classe DicomImageReader

```

...
037 DicomInputStream inputStream = new DicomInputStream(file);
038 DicomObject dicomObj = inputStream.readDicomObject();
040 image = new DicomImage();
041 double spacingBetweenSlices =
    dicomObj.getDouble(Tag.SpacingBetweenSlices);

042 dicom.setSpacingBetweenSlices(spacingBetweenSlices);
043 image.setBitsAllocated(dicomObj.getInt(Tag.BitsAllocated));
044 image.setPixelRepresentation(
    dicomObj.getInt(Tag.PixelRepresentation));

045 image.setColumns(dicomObj.getInt(Tag.Columns));
046 image.setImageType(dicomObj.getString(Tag.ImageType));
047 image.setBigEndian(dicomObj.bigEndian());
...

```

Possuindo a informação de quantos `bits` são alocados para a imagem, podemos fazer a conversão necessária para ser possível trabalhar com os canais *Red Green Blue Alpha* (RGBA) da imagem. Foi implementando num primeiro momento, apenas a conversão para imagens com 16 `bits` alocados. O método `readImageDataSet` que verifica os `bits` alocados para a imagem, e chama o método adequado para cada tipo de formato. Como foi implementado apenas o suporte para imagens de 16 `bits`, o método `read16BitsImage` é chamado, onde nele acontece a conversão da imagem para um `array` de inteiros utilizado no componente `Bitmap` do Android.

Em um primeiro momento, é criado um `array` de inteiros bidimensional, onde serão armazenados os `pixels` em forma de matriz. Para cada `pixel`, é verificado o valor do `pixel` comparando com um limiar pré-definido, com valor 50, e então é trabalhado os quatro canais do `pixel`, sendo os três primeiros canais as cores, representando respectivamente vermelho (Red), verde (Green) e azul (Blue) e o último canal, o valor `alpha` do `pixel`, representando se aquele `pixel` possui ou não transparência. No Quadros 7 e 8 temos respectivamente o trecho do método `read16BitsImage` e o método `limiar`.

Quadro 7 - Método read16BitsImage da classe DicomImageReader

```

137 int totBytes = pixels.length;
138 int shortSize = totBytes / 2;
141 double minimum = dicomObject.getDouble(Tag.WindowCenter) -
    dicomObject.getDouble(Tag.WindowWidth) / 2;
142 double maximum = dicomObject.getDouble(Tag.WindowCenter) +
    dicomObject.getDouble(Tag.WindowWidth) / 2;
143 if (minimum < 0.0)
144     minimum = 0.0;
145 if (maximum > 65535.0)
146     maximum = 65535.0;
147 int min = (int) minimum;
148 int max = (int) maximum;
150 int value;
151 int x = 0;
152 int y = 0;
153 int limiar = 0;
154 double scale = 256.0 / (max - min + 1);
156 matrix = new int[image.getColumns()][image.getRows()];
158 short shortValue = 0;
160 for (int i = 0; i < shortSize; i++) {
161     shortValue = (short) (((pixels[2 * i + 1] & 0xFF) << 8) |
        (pixels[2 * i] & 0xFF));
163     value = (shortValue & 0xffff) - min;
164     if (value < 0)
165         value = 0;
166     value = (int) (value * scale + 0.5);
167     if (value > 255)
168         value = 255;
170     limiar = limiar(value);
172     matrix[x][y] += limiar << 24;
173     matrix[x][y] += value << 16;
174     matrix[x][y] += value << 8;
175     matrix[x][y] += value;
177     if(++x == image.getColumns()) {
178         x = 0;
178         y++;
180     }
181 }

```

Quadro 8 - Método limiar da classe DicomImageReader

```
130 if(pixel < LIMIAR) {  
131     return 0;  
142 } else {  
143     return 255;  
144 }
```

Para reduzir o custo de processamento e memória do aplicativo, foi implementado o conceito de `bounding-box` nas imagens. A `bounding-box` delimita a área útil de uma determinada região. No caso das imagens DICOM, essa área útil, é delimitada por uma região que contenha ao menos um `pixel` que seja diferente do valor aplicado aos `pixels` com transparência, neste caso representando um `pixel` que não possui uma informação e que consequentemente não precisa ser desenhada. Então, considerando que estamos representando nossos `pixels` em uma matriz, começa-se a varredura na primeira coluna da matriz, percorrendo todas as linhas daquela coluna, verificando se há algum `pixel` que não seja transparente. Caso nada seja encontrado, o valor de `x` que representa nossa coluna, é incrementado. Esse processo é feito até o algoritmo localizar um `pixel` não transparente, e então o valor de `x` atual é definido como `x mínimo`. O mesmo processo é feito para encontrar o `x máximo`, mas de maneira inversa, começando a última coluna e decrementando o valor de `x`. O mesmo processo é feito para os valores de `y mínimo` e `y máximo`. Os Quadros 9, 10, 11 e 12 demonstram o algoritmo dos métodos para encontrar os valores máximos e mínimos de `x` e `y`.

Quadro 9 - Método findMaxY da classe DicomImageReader

```

069 private int findMaxY() {
070     for (int y = matrix[0].length - 1; y >= 0; y--) {
071         for (int x = 0; x < matrix.length; x++) {
072             if (matrix[x][y] != 0) {
073                 maxY = y;
074                 return maxY;
075             }
076         }
077     }
078 }

```

Quadro 10 - Método findMinY da classe DicomImageReader

```

081 private int findMinY() {
082     for (int y = 0; y < matrix[0].length; y++) {
083         for (int x = 0; x < matrix.length; x++) {
084             if(matrix[x][y] != 0) {
085                 minY = y;
086                 return minY;
087             }
088         }
088     }
089 }

```

Quadro 11 - Método findMaxX da classe DicomImageReader

```

093 private int findMaxX() {
094     for (int i = matrix.length - 1; i >= 0; i--) {
095         for (int j = 0; j < matrix[i].length; j++) {
096             if(matrix[i][j] != 0) {
097                 maxX = i;
098                 return maxX;
099             }
100         }
101     }
102 }

```

Quadro 12 - Método findMinX da classe DicomImageReader

```

105 private int findMinX() {
106     for (int i = 0; i < matrix.length; i++) {
107         for (int j = 0; j < matrix[i].length; j++) {
108             if(matrix[i][j] != 0) {
109                 minX = i;
110                 Return minX;
111             }

```

```

112     }
113 }
114 }

```

Encontrados os valores da `bounding-box` de cada imagem, eles são armazenados no objeto da classe `DicomImage` de cada uma das imagens, para posteriormente na classe `DicomReader`, todas imagens serem analisadas e é verificado dentre todas elas, qual o menor `x` mínimo, o menor `y` mínimo, o maior `x` máximo e o maior `y` máximo. De posse desses valores, podemos desconsiderar os `pixels` que estão fora da nossa `bounding-box`. O Quadro 13 demonstra esse processo.

Quadro 13 - Trecho de código do método `readImages` da classe `DicomReader`

```

086 int minX = Integer.MAX_VALUE;
087 int minY = Integer.MAX_VALUE;
088 int maxX = 0;
089 int maxY = 0;
090 for (DicomImage image : images) {
091     if(image.getMinX() < minX) {
092         minX = image.getMinX();
093     }
094     if(image.getMinY() < minY) {
095         minY = image.getMinY();
096     }
097     if(image.getMaxX() > maxX) {
098         maxX = image.getMaxX();
099     }
100     if(image.getMaxY() > maxY) {
101         maxY = image.getMaxY();
102     }
103 }
105 for (DicomImage image : images) {
106     image.applyBoundingBox(minX, maxX, minY, maxY);
107 }

```

Nesse momento, o processo se resume e criar uma nova matriz com o tamanho relativo aos novos valores de `x` e `y`. Ou seja, a largura da matriz passa a ser `x` máximo subtraído ao `x` mínimo somando um. O Quadro 14 apresenta em detalhes o código dessa função.

Quadro 14 - Método applyBoundingBox da classe DicomImage

```

152 public void applyBoundingBox(int minX, int maxX, int minY, int
    maxY) {
153     int columns = maxX - minX + 1;
154     int rows = maxY - minY + 1;
155     setColumns(columns);
156     setRows(rows);
157     int x, xOriginal, yOriginal;
158     int y = 0;
159     int[] pixels = new int[rows * columns];
160     int[][] newMatrix = new int[columns][rows];
162     int count = 0;
164     for (int i = 0; i < columns; i++) {
165         for (int j = 0; j < rows; j++) {
166             y = count / columns;
167             x = count - (y * columns);
168             xOriginal = x + minX;
169             yOriginal = y + minY;
171             newMatrix[i][j] = matrix[xOriginal][yOriginal];
172             pixels[count] = matrix[xOriginal][yOriginal];
173             count++;
174         }
175     }
177     matrix = newMatrix;
178     pixelData = pixels;
179 }

```

Um novo array de inteiros foi necessário porque o componente `Bitmap` do Android aceita apenas array de inteiros para a criação da imagem.

3.3.2.2 Visualização em duas dimensões

O processo de visualização em duas dimensões se dá através da classe `ViewerActivity`, onde tem-se quatro quadros, sendo que nos três primeiros são apresentadas as imagens em duas dimensões nas direções sagital, coronal e axial respectivamente. E no quarto quadro, apresenta-se o botão que chamará a tela de visualização volumétrica.

O primeiro quadro da tela apresentará as imagens na direção sagital, ou seja, as imagens não precisarão sofrer nenhuma manipulação pelo fato de que o exame já provém as imagens originalmente nessa direção. A única implementação foi controlar a imagem exibida de acordo com a posição do `slider` do quadro Sagital.

O segundo quadro da tela apresenta as imagens na direção coronal. Para conseguir gerar as imagens nessa direção, foi necessário capturar todas colunas na posição k de todas as imagens, sendo k o índice selecionado no `slider`, variando de zero até o número máximo de colunas das imagens. Como o número de imagens é geralmente muito menor que a largura das imagens, utilizou-se um valor de compensação para cada imagem. Ou seja, para cada coluna capturada de cada imagem, ela foi multiplicada por cinco para fazer com que a largura da imagem seja proporcional a altura dela. Então caso tenha-se cinquenta e cinco imagens de 512×512 , teria-se na direção coronal, imagens de 275 (55 imagens multiplicadas pelo quociente definido, de valor 5) $\times 512$. O Quadro 15 apresenta o algoritmo que cria a imagem na direção coronal.

Quadro 15 - Método `setImage` da classe `CoronalContainer`

```

033 private void setImage(int index) {
034     List<DicomImage> images = getDicom().getImages();
036     final int CONST = 5;
037     int[] pixels = new
    int[getDicom().getImages().get(0).getRows() *
    getDicom().getImages().size() * CONST];
038     int count = 0;
040     int rows = getDicom().getImages().get(0).getMatrix().length;
041     for (int row = 0; row < rows; row++) {
042         for (int i = 0; i < images.size(); i++) {
043             DicomImage image = images.get(i);
044             int[][] matrix = image.getMatrix();
045             for (int j = 0; j < CONST; j++) {
046                 pixels[count++] = matrix[row][index];
047             }
048         }
049     }
051     DicomImage image = new DicomImage();
052     image.setColumns(getDicom().getImages().size() * CONST);
053     image.setRows(rows);
055     Bitmap bitmap = image.createBitmap(pixels);
056     imageView.setImageBitmap(bitmap);
    ...
062 }

```

O terceiro quadro da tela apresenta as imagens na direção axial. Para conseguir gerar as imagens na direção axial, foi utilizado o mesmo procedimento da geração das imagens na direção coronal, diferenciando apenas a captura de colunas na direção coronal, e agora na

axial, utilizamos as linhas de todas as imagens. O processo de compensação de ocorre nas imagens na direção axial da mesma forma, alterando apenas colunas por linhas. Se na direção coronal as colunas eram multiplicadas pelo quociente definido, agora quem serão multiplicadas serão as linhas. O Quadro 16 apresenta o algoritmo de geração de imagens na direção axial.

Quadro 16 - Método setImage da classe AxialContainer

```

033 private void setImage(int index) {
034     List<DicomImage> images = getDicom().getImages();
036     final int CONST = 5;
037     int[] pixels = new
int[getDicom().getImages().get(0).getColumns() *
getDicom().getImages().size() * CONST];
038     int count = 0;
040     for (int i = images.size() - 1; i >= 0; i--) {
041         DicomImage image = images.get(i);
042         int[] pixelData = image.getPixelData();
044         int indexIni = index * image.getColumns();
045         for (int k = 0; k < CONST; k++) {
046             for (int j = 0; j < image.getColumns(); j++) {
047                 pixels[count++] = pixelData[indexIni + j];
048             }
049         }
050     }
052     DicomImage image = new DicomImage();
053     image.setColumns(
getDicom().getImages().get(0).getColumns());
054     image.setRows(getDicom().getImages().size() * CONST);
056     Bitmap bitmap = image.createBitmap(pixels);
057     imageView.setImageBitmap(bitmap);
    ...
063 }

```

3.3.2.3 Visualização volumétrica

O processo de visualização volumétrica se dá início logo após a leitura das imagens. A classe `VolumetricViewerRenderer` inicia o processo de visualização volumétrica do exame, primeiramente chamando o método `onSurfaceCreated`, onde basicamente carregamos as

texturas e configuramos as variáveis necessárias para o uso de texturas no OpenGL. O Quadro 17 apresenta a forma que as texturas estão sendo carregadas.

Posterior a isso, o OpenGL se encarrega de chamar constantemente o método `onDrawFrame`. Nesse método a cena é limpa e a câmera é posicionada de acordo com os valores do atributo `camera` da classe `VolumetricViewerRenderer`.

Depois de posicionado a câmera, o método `draw` da classe `Square` é chamado. Este método é encarregado de desenhar, para cada imagem, uma superfície quadrada, respeitando a distância definida pela tag `SpacingBetweenSlices` do arquivo DICOM. Posteriormente, a textura de cada imagem é aplicada a cada superfície quadrada desenhado na cena, formando assim, o objeto volumétrico. O Quadro 18 apresenta o código responsável por desenhar as superfícies quadradas na tela e aplicar as texturas à eles.

Quadro 17 - Método loadGLTextures da classe Square

```

108 public void loadGLTextures(GL10 gl) {
109     Bitmap bitmap = null;
110     gl.glGenTextures(textures.length, textures, 0);
111     for (int i = 0; i < textures.length; i++) {
112         gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[i]);
113         gl.glTexParameterf(GL10.GL_TEXTURE_2D,
114             GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
115         gl.glTexParameterf(GL10.GL_TEXTURE_2D,
116             GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
117         gl.glTexParameterf(GL10.GL_TEXTURE_2D,
118             GL10.GL_TEXTURE_WRAP_S, GL10.GL_REPEAT);
119         gl.glTexParameterf(GL10.GL_TEXTURE_2D,
120             GL10.GL_TEXTURE_WRAP_T, GL10.GL_REPEAT);
121         bitmap = bitmaps.get(i);
122         GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
123         gl.glBindTexture(GL10.GL_TEXTURE_2D, 0);
124     }
125 }

```

Quadro 18 - Método draw da classe Square

```

072 public void draw(GL10 gl) {
073     gl.glMatrixMode(GL10.GL_MODELVIEW);
074     float z = 0.0078125f;
075     gl.glEnable(GL10.GL_TEXTURE_2D);
076     gl.glDepthMask(false);
077     gl.glEnable(GL10.GL_BLEND);
078     gl.glBlendFunc(GL10.GL_SRC_ALPHA,
079         GL10.GL_ONE_MINUS_SRC_ALPHA);
080     gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
081     gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
082     for (int i = 0; i < textures.length; i++) {
083         gl.glTranslatef(0.0f, 0.0f, z);
084         gl.glBindTexture(GL10.GL_TEXTURE_2D, textures[i]);
085         gl.glFrontFace(GL10.GL_CW);
086         gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
087         gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0,
088             textureBuffer);
089         gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0,
090             vertices.length / 3);
091     }
092     gl.glDisable(GL10.GL_BLEND);
093 }

```

```
100    gl.glDepthMask(true);
102    gl.glDisable(GL10.GL_TEXTURE_2D);
104    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
105    gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
106 }
```



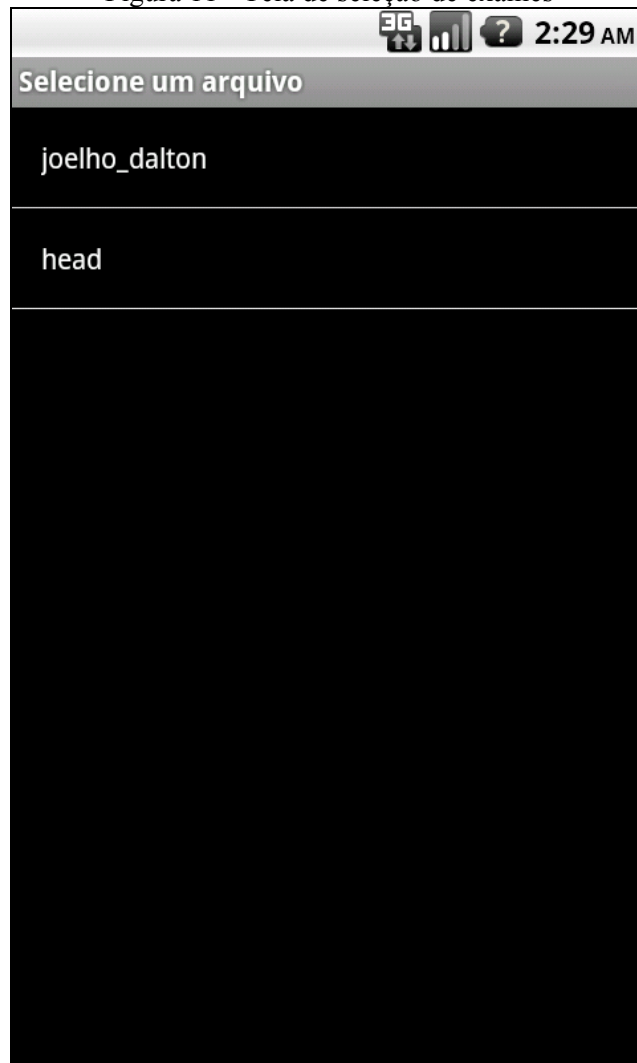
3.3.3 Operacionalidade da implementação

Nesta seção é apresentada a utilização da aplicação. São demonstrados o funcionamento das telas e usabilidade. Primeiramente, a tela de visualização em duas dimensões seguido da tela de visualização volumétrica.

3.3.3.1 Visualização em duas dimensões

Ao usuário executar a aplicação, a tela de `seleção do exame` é aberta, como podemos ver na Figura 11. Selecionado um exame, os arquivos de informação do exame bem como suas imagens, são lidos para posteriormente serem usados tanto na visualização em duas dimensões quanto na visualização volumétrica.

Figura 11 - Tela de seleção de exames



Depois de selecionado um exame na tela de seleção de exames, a tela de visualização em duas dimensões é aberta, dividindo a tela em quatro partes distintas. Os três primeiros quadros apresentam a visualização em duas dimensões em diferentes direções. Esses quadros possuem três componentes, sendo o primeiro um componente de texto com o título informando o modo de visualização da imagem, a posição da imagem atual e o número máximo de imagens à serem exibidas naquela direção. Além disso, tem-se um `slider` usado para a alteração da imagem visualizada e o componente para a visualização da imagem atual. A Figura 12 ilustra essa tela.

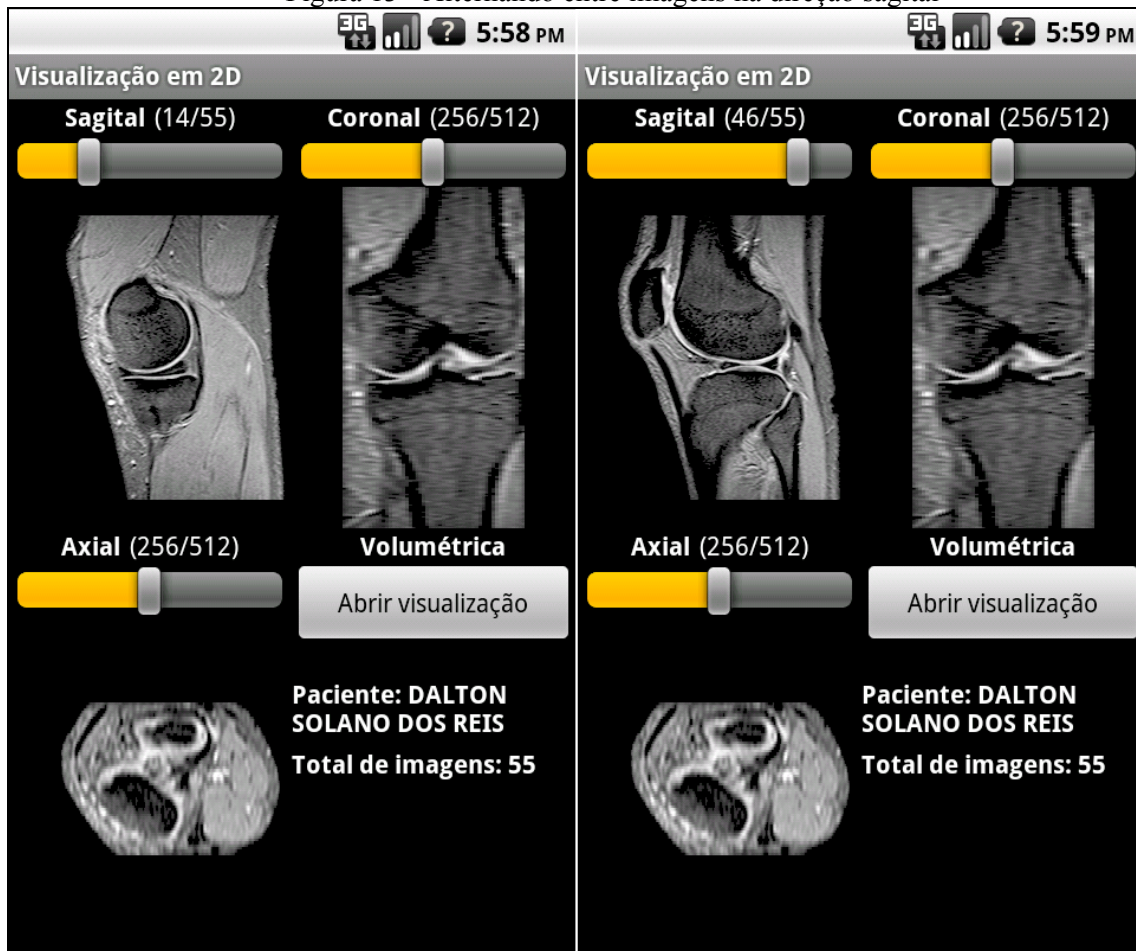
Figura 12 - Tela de visualização em duas dimensões



O primeiro quadro apresenta as imagens na direção sagital, como mostra a Figura 12. O componente `slider`, nesse quadro, tem uma variação de zero até o número máximo de imagens do exame selecionado. Inicialmente, o `slider` vem com sua posição definida exatamente no centro. Então ao abrir a tela de visualização em duas dimensões, teremos no quadro sagital, a imagem correspondente a $n/2$ do exame exibida, sendo n o número de imagens do exame selecionado. Para navegar entre as imagens do exame na direção sagital, basta modificar a posição do `slider`, que a imagem correspondente a posição atual do slider é exibida.

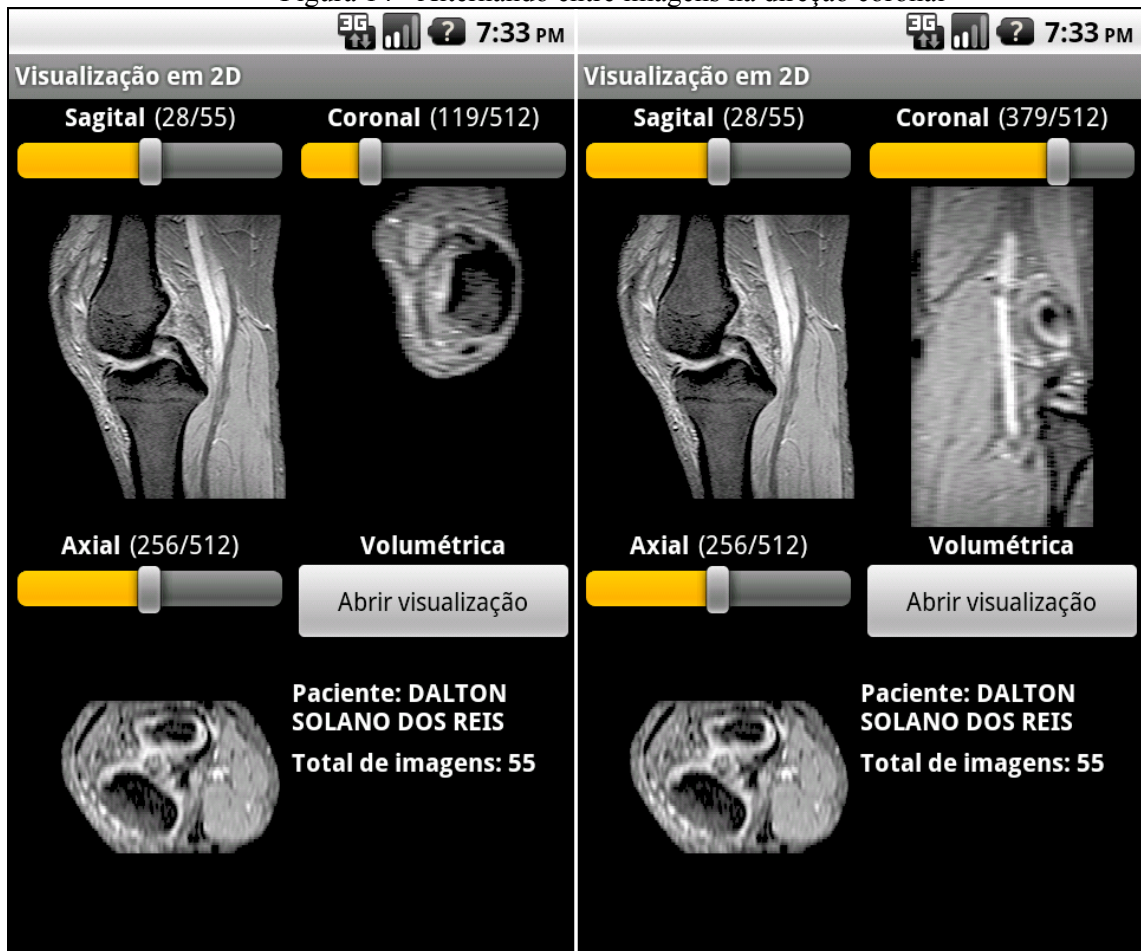
Ao modificar a posição do `slider`, a imagem selecionada é alterada, como podemos ver na Figura 13.

Figura 13 - Alternando entre imagens na direção sagital



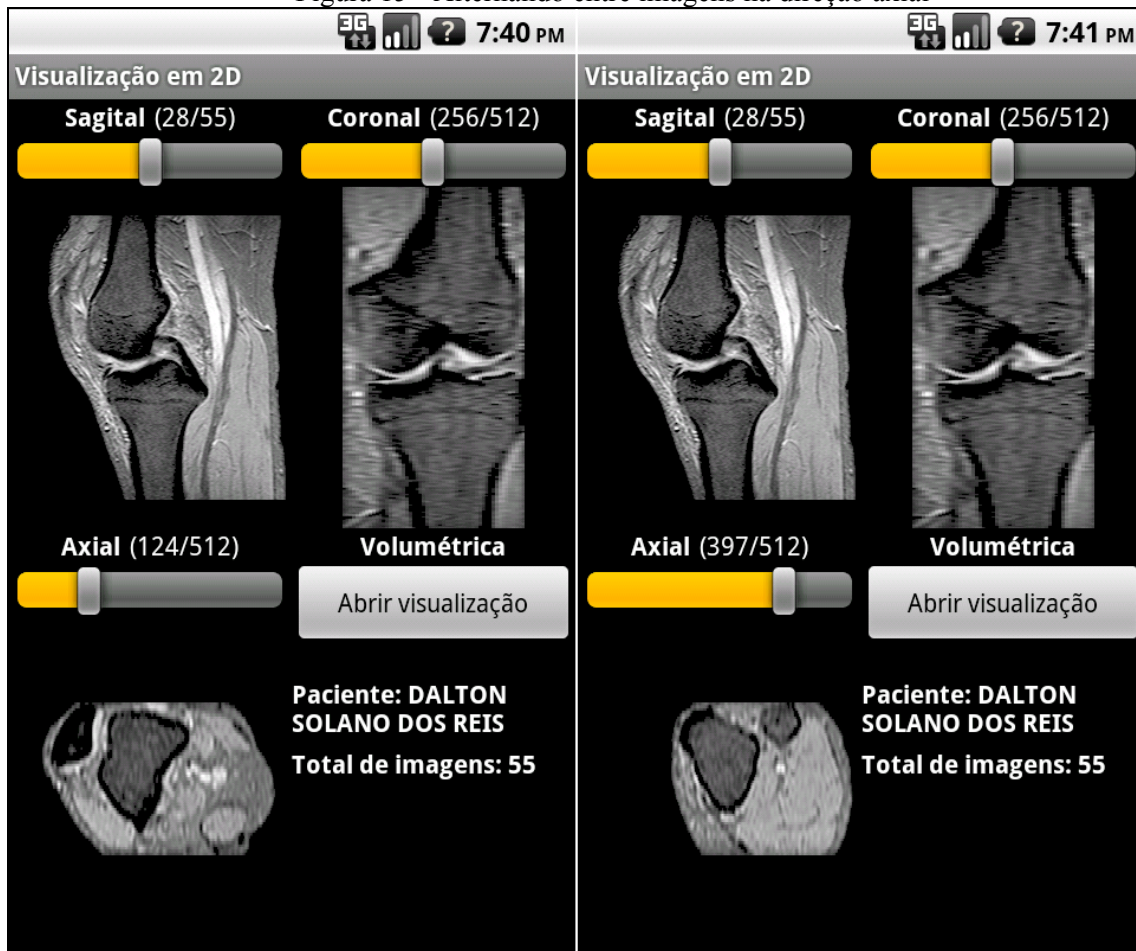
O segundo quadro apresenta as imagens na direção coronal. Nesse quadro, o `slider` varia de zero até n , sendo n a largura das imagens. Por padrão, a imagem da posição $n/2$ é exibida, como nos outros quadros. Para alterar a imagem atual visualizada, basta mover a posição atual do `slider` para a nova posição desejada, como é demonstrado na Figura 14.

Figura 14 - Alternando entre imagens na direção coronal



O terceiro quadro apresenta as imagens na direção axial. Nesse quadro, o `slider` varia de zero até n , sendo n a altura das imagens. Assim como nos outros quadros, para alterar a imagem visualizada, basta alterar a posição do `slider`, como mostra a Figura 15.

Figura 15 - Alternando entre imagens na direção axial



Por último, temos o quadro onde temos o botão para a chamada da tela de visualização volumétrica e as informações básicas como nome do paciente e o número de imagens do exame em questão.

3.3.3.2 Visualização volumétrica

Ao selecionar o botão `Abrir Visualização` no último quadro da tela de visualização em duas dimensões, a tela de visualização volumétrica é aberta. Nela é feita a renderização de todas as imagens lidas do exame, e posicionadas para que formem a visualização volumétrica. Para voltar para a tela de visualização em duas dimensões, basta selecionar o botão `Voltar` do próprio dispositivo usado. A Figura 16 apresenta a tela de visualização volumétrica.

Figura 16 - Tela de visualização volumétrica



3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta uma aplicação para a visualização volumétrica de imagens DICOM para a plataforma Android. Nesta etapa inicialmente são mostrados os resultados do volume gerado a partir das imagens recuperadas dos arquivos lidos. Logo a seguir são comparados os resultados deste trabalho com o trabalho correlato, visto que o primeiro trabalho correlato apresentado não teve o desenvolvimento de uma aplicação, apenas um estudo teórico sobre o tema. Por fim é discutido o consumo de memória da aplicação e seu desempenho.

3.4.1 Testes da geração do volume

Na etapa de leitura dos arquivos, pesquisou-se várias alternativas que tivessem suporte a plataforma Android. A única alternativa viável foi utilizar a biblioteca DCM4CHE, que por padrão foi desenvolvida para rodar em ambientes *Java Enterprise Edition* (JEE). Como a

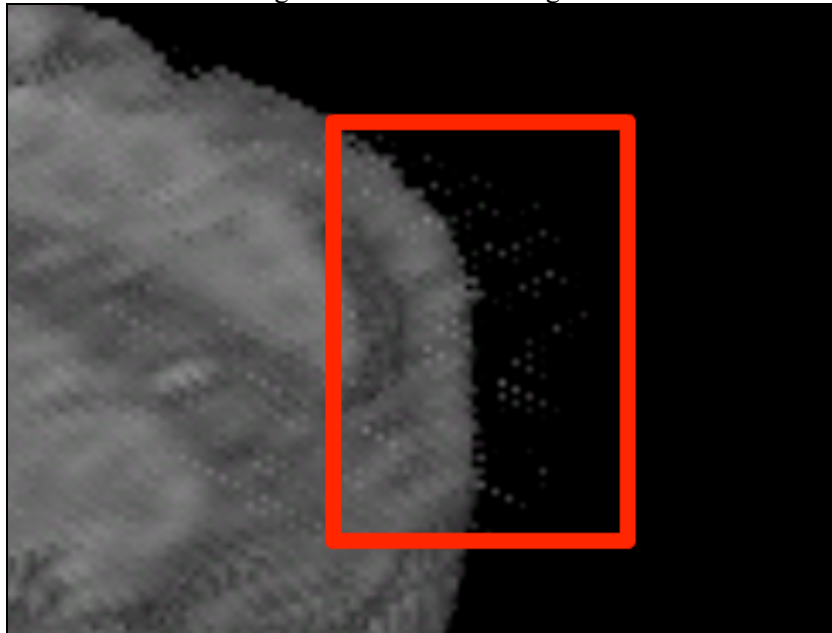
biblioteca pesquisada é subdividida em outras pequenas bibliotecas, foi realizado um estudo de quais partes seriam necessárias para a leitura dos exames. Feito isso, as partes necessárias foram incluídas no projeto e então foram realizados os testes de leitura, que logo apresentaram resultados satisfatórios na quantidade de informação que a biblioteca realizava a leitura. Porém, o desempenho obtido no processo de leitura foi um pouco abaixo do esperado. Mais detalhes sobre o desempenho são apresentados na seção 3.4.3.

Nos arquivos de imagem, a biblioteca consegue ler os `pixels` da imagem em forma de `byte array` através da `tag PixelData`. Na aplicação apresentada, limitou-se a leitura das imagens apenas para as que tenham representação em dezesseis `bits` com ou sem sinal. Tendo em mãos o `array de bytes`, o próximo passo foi convertê-lo para o formato adequado a imagem lida, que no caso da aplicação apresentada, limitou-se a dezesseis `bits`. Os `pixels` convertidos foram armazenados em uma matriz de inteiros, com cada posição da matriz representando um `pixel` da imagem. No mesmo momento do processo de conversão para a representação de `bits` adequada, é verificado a área útil da imagem, ou seja, a área que possui um `pixel` que não seja a cor de fundo da imagem. Para isso, utilizou-se um limiar, com valor fixo, que a cada leitura de `pixel` da imagem, verifica se o `pixel` lido corresponde ao fundo da imagem, comparando o valor dele com o limiar definido. Caso o `pixel` seja menor que o valor do limiar, aplica-se ao canal `alpha` do `pixel` o valor máximo, para que no momento em que o OpenGL-ES desenhar essa imagem na cena, os `pixels` com valor `alpha` máximo sejam desenhados com valor mínimo de opacidade e assim, não apareçam na cena.

Logo após a conversão e aplicação do limiar, é aplicado na matriz obtida, o conceito de `bounding-box`, verificando em cada imagem, qual a coluna e linha onde se inicia o corpo de cada imagem. Isso é possível comparando a menor e maior linha e coluna que possuem o valor mínimo, ou seja, zero, no canal `alpha` do `pixel`. Este processo é realizado em todas as imagens, e depois de ter lido todas as imagens, verifica-se a maior e menor linha e coluna, e então é criada uma nova matriz apenas com as linhas e colunas que fazem parte do corpo da imagem.

Foi esperado um resultado mais satisfatório com a utilização da `bounding-box`, visto que no exame do joelho, o tamanho das imagens diminuiu em média duas colunas. Não houve tempo hábil para verificar esse quesito, mas acredita-se que ruídos nas imagens podem ter causado esse resultado não satisfatório, como podemos ver mais nitidamente na Figura 17.

Figura 17 - Ruído na imagem



Nos testes utilizando o dispositivo, foi encontrado uma limitação na utilização das texturas no OpenGL-ES, que aceita apenas imagens que tenham a mesma altura e largura. Isso inviabilizou o uso da *bounding-box* nos testes no dispositivo utilizado. Por esse motivo, nos testes no dispositivo, a *bounding-box* foi desabilitada.

Ainda na visualização volumétrica, não obteve-se um resultado satisfatório na rotação do objeto, tendo como problema a sobreposição do objeto na nova posição em relação ao objeto na posição anterior, e assim sucessivamente. Mesmo utilizando os comandos do OpenGL-ES para apagar a cena a cada nova mudança de posição da câmera, o problema persistia. Devido ao tempo, não foi possível resolver este problema, mas o código da aplicação está preparado para receber a rotação do objeto, precisando apenas identificar qual o real problema em apagar o *buffer* de renderização da cena a cada nova rotação.

3.4.2 Comparação com o trabalho correlato

Nesta seção são comparados as principais características deste trabalho com as do trabalho correlato. A Tabela 1 apresenta um comparativo entre este trabalho e o trabalho correlato, Osirix.

Tabela 1 - Comparação entre este trabalho com o trabalho correlato

Características	Osirix	Hermann (2013)
Ler arquivos DICOM	X	X
Visualização em duas dimensões	X	X
Tratamento das imagens	X	
Visualização volumétrica	X	X
Rotação do volume	X	
Visualização interna do volume	X	
Visualização em mais de três dimensões	X	
Separação do volume em componentes	X	
Fatiamento do volume	X	
Geração da imagem em mais de uma direção anatômica	X	X
Aplicação para dispositivos móveis	X	X

Observando a tabela, percebe-se que o Osirix é uma aplicação muito mais completa, possuindo todas as características observadas. Segundo Osirix (2013), a aplicação foi resultado de mais de cinco anos de pesquisa e desenvolvimento em imagens digitais.

A maior dificuldade ao comparar os resultados, foi encontrar um aplicativo na plataforma Android que realizasse a visualização volumétrica ou gerasse as imagens em outras direções além da sagital. Os aplicativos encontrados, realizavam apenas a visualização das imagens na direção sagital, por este motivo, não foi comparado nenhum deles com o trabalho apresentado.

3.4.3 Memória e desempenho

Para realizar os testes de desempenho e memória da aplicação foram utilizados dois exames médicos obtidos por ressonância magnética. O primeiro exame utilizado foi do crânio, que possui 22 imagens, sendo que cada uma das imagens possui tamanho 256x256. O segundo exame, foi o do joelho, tendo 55 imagens de tamanho 512x512. A aplicação foi executada utilizando o simulador do Android na versão 2.2 e o tablet Samsung Galaxy Tab 2 com processador de 1 GHz e memória de 1 GB, rodando Android na versão 4.1.1. Os testes utilizando todas as imagens foram executadas somente no simulador, já que utilizando o dispositivo, no exame do joelho, o uso de memória excedia o limite máximo disponível para a aplicação. Como no dispositivo não temos como aumentar o valor máximo de memória

disponível, foi necessário executar utilizando o simulador, que permite esse aumento necessário. Os testes no dispositivo limitaram a leitura de imagens para 10 unidades e todos resultados apresentados referentes ao dispositivo, consideram apenas 10 imagens.

Para a análise do consumo de memória, foi utilizado a ferramenta Heap que está inclusa no *Software Development Kit* (SDK) do Android.

3.4.3.1 Consumo de memória

Será analisado nesta seção, o consumo de memória dos dois exames em quatro momentos distintos da aplicação. Primeiramente, o uso de memória quando a aplicação é iniciada, quando a tela de seleção de exames é exibida. Logo em seguida, é verificado o consumo de memória da leitura do arquivo DICOM assim como suas imagens, seguido da visualização em duas dimensões e da visualização volumétrica. A Tabela 2 e 3 apresentam os resultados obtidos nos diferentes exames, no simulador e no dispositivo, respectivamente. Uma ilustração melhor desta tabela pode ser observado na Figura 18 e 19.

Tabela 2 - Consumo de memória da aplicação utilizando o simulador

Momento	Crânio	Joelho
Início da aplicação	2,277 MB	2,277 MB
Leitura dos arquivos	9,442 MB	113,100 MB
Visualização 2D	9,484 MB	113,140 MB
Visualização volumétrica	9,508 MB	113,147 MB

Tabela 3 - Consumo de memória da aplicação utilizando o dispositivo

Momento	Crânio	Joelho
Início da aplicação	6,927 MB	6,927 MB
Leitura dos arquivos	12,563 MB	28,131 MB
Visualização 2D	12,480 MB	27,396 MB
Visualização volumétrica	14,763 MB	37,428 MB

Figura 18 - Gráfico do uso de memória utilizando o simulador

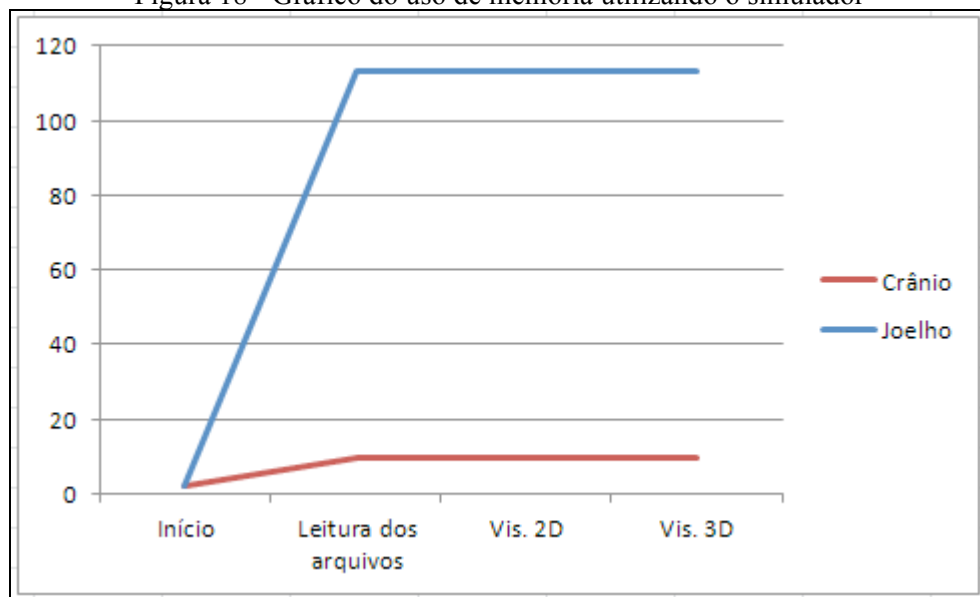
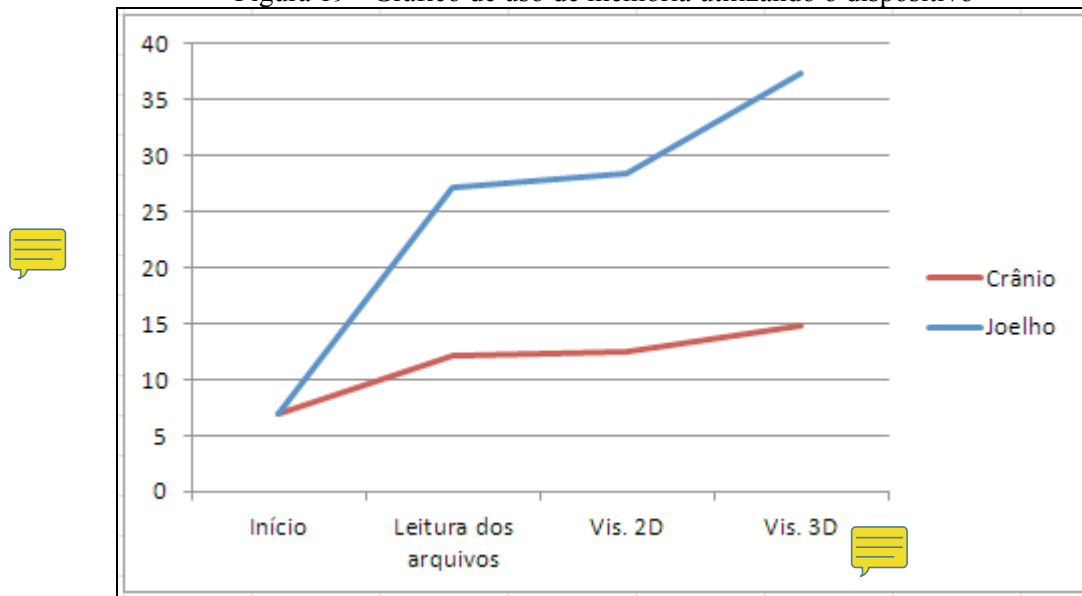


Figura 19 - Gráfico de uso de memória utilizando o dispositivo



Observa-se que o uso maior de memória ocorre no momento da leitura dos arquivos. A diferença fica mais evidente no exame do joelho, devido ao número de imagens e a dimensão das mesmas.

Percebe-se uma diminuição mínima no consumo de memória após a leitura dos arquivos, quando se dá início a visualização em duas dimensões e posteriormente a volumétrica. Isso se deve ao fato que a máquina virtual do Android possui um mecanismo de limpeza de lixo, mais conhecido como *Garbage Collector* (GC), que faz uma varredura nos objetos instanciados e elimina os que não estão mais sendo usados.

A diferença mais significativa percebida entre os testes no simulador e no dispositivo, foram na visualização volumétrica, mas não foi encontrado nenhuma razão comprovada para esta diferença.

Em uma futura melhoria do aplicativo, o consumo de memória deve ser um dos primeiros itens analisados para tornar possível a execução do mesmo nos dispositivos com uma capacidade de memória razoável e possibilitar a leitura de qualquer exame independente da quantidade e tamanho de imagem utilizados.

3.4.3.2 Desempenho

A avaliação do desempenho seguiu a mesma linha, comparando os dois exames e verificando o tempo gasto em operações específicas do aplicativo.

Nos testes feitos usando o simulador, não deve-se levar em consideração que os tempos apresentados são tempos reais de execução em um dispositivo, já que no simulador, toda operação é mais custosa e consequentemente, demanda mais tempo. Nos testes

realizados no dispositivo, foram considerados apenas a leitura e exibição de 10 imagens, como já explicado na seção 3.4.3. A Tabela 4 e 5 apresentam os resultados obtidos em diferentes operações da aplicação utilizando o simulador e o dispositivo, respectivamente.

Tabela 4 - Desempenho por operação utilizando o simulador

Operação	Crânio	Joelho
Leitura dos arquivos	12,825 s	136,369 s
Criação da imagem coronal	0,079 s	0,291 s
Criação da imagem axial	0,073 s	0,236 s
Montagem da tela de visualização 2D	0,317 s	0,985 s

Tabela 5 - Desempenho por operação utilizando o dispositivo

Operação	Crânio	Joelho
Leitura dos arquivos	0,630 s	2,333 s
Criação da imagem coronal	0,153 s	0,212 s
Criação da imagem axial	0,055 s	0,056 s
Montagem da tela de visualização 2D	0,049 s	0,057 s

Observa-se que o processo mais custoso foi o da leitura dos arquivos, sendo que no exame do joelho, onde temos 55 imagens de tamanho 512x512, o tempo gasto ultrapassou 2 minutos utilizando o simulador, que não é uma situação aceitável para uma aplicação final. Já lendo as 10 imagens do exame do joelho no dispositivo, obteve-se um tempo de pouco mais de 2 segundos, que resultaria no tempo médio de 12 segundos para ler as 55 imagens do exame.

Uma otimização na leitura dos arquivos, talvez até criando um leitor DICOM próprio, seria de extrema importância numa futura evolução do aplicativo.

Detalhando um pouco mais os tempos gastos na leitura das imagens, percebeu-se que o maior tempo ocupado é na conversão dos `bytes` obtidos pela leitor para a lista de inteiros representados em 16 `bits`, apresentado no Quadro 7.



4 CONCLUSÕES

Neste trabalho apresentou-se o desenvolvimento de uma aplicação de visualização volumétrica de imagens DICOM na plataforma Android. Os resultados dos testes realizados mostram que o aplicativo atende parcialmente os objetivos propostos.

Este trabalho teve contribuição importante com o desenvolvimento de um aplicativo que fosse possível a leitura das imagens e exibição das mesmas em três direções distintas. Além disso, apresentou a visualização volumétrica, mesmo que numa qualidade mediana e sem proporcionar a rotação e fatiamento do volume.

A geração da visualização volumétrica foi possível utilizando várias fatias no espaço, respeitando uma distância pré-determinada entre elas, e em cada uma das fatias, foi aplicada a textura da imagem correspondente, já com sua transparência aplicada.

A qualidade do volume gerado volumetricamente, é diretamente proporcional a quantidade de imagens no exame. Quanto maior o número de imagens, maior será a qualidade, pois não foi implementado nenhum mecanismo que ocupasse a área vazia entre cada uma das fatias.

O maior desafio no desenvolvimento do aplicativo foi o entendimento de como eram feitas as conversões das imagens lidas do arquivo DICOM para o formato que fosse possível ser lido no componente de imagem do Android. Além da dificuldade relatada, a manipulação da câmera juntamente com texturas não foi totalmente satisfatória, apresentando o problema de, ao rotacionar o volume em uma nova posição, ele era desenhado nessa nova posição, mas o volume na posição anterior era mantido, mesmo utilizando todos os comandos para limpar a cena.

4.1 EXTENSÕES

Após o desenvolvimento deste trabalho, pode-se indentificar alguns pontos de melhoria, que são:

- a) otimização no processo de leitura dos arquivos DICOM, incluindo a possibilidade de ler imagens além de 16 bits;
- b) redução do uso de memória no processo de leitura dos arquivos DICOM, para que seja possível realizar a leitura de qualquer exame em um dispositivo físico;
- c) realizar a rotação do volume;
- d) realizar o fatiamento do volume nas três direções;
- e) desenhar a superfície do volume, tomando como referência os pontos externos de cada imagem, e assim realizar o desenho do corpo externo do volume.

REFERÊNCIAS BIBLIOGRÁFICAS

- ANDROID DEVELOPERS. [S.l.], 2009. Disponível em: <<http://developer.android.com/about/index.html>>. Acesso em: 20 set. 2013.
- APPLE ITUNES. [S.l.], 2012. Disponível em: <<http://www.apple.com/itunes/>>. Acesso em: 20 set. 2013.
- CARNEIRO, Marcelo M.; MARTHA, Luiz F. C. R. **Visualização interativa 3D de dados volumétricos**. 2000. 47 p. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.
- GOOGLE. [S.l.], 2013. Disponível em: <<http://www.google.com.br/about/company/>>. Acesso em: 20 set. 2013.
- LECHETA, Ricardo. **Google Android**: aprenda a criar aplicações para dispositivos móveis com o Android SDK. 2. edição. Novatec, 2010. 608 p.
- MCCORMICK, Bruce; DEFANTI, Thomas; BROWN, Maxine. (1987). **Visualization in Scientific Computing**. *SIGBIO ACM Special Interest Group on Biomedical Computing*, New York, v.10, p 15-21, 1987.
- MONTEIRO, Denyse N. B. **Estudo sobre a visualização de imagens médicas obtida por exames virtuais**. 2005. 121 f. Dissertação (Mestrado em Computação) - Curso de Pós-graduação em Computação, Universidade Federal Fluminense, Niterói.
- OSIRIX. [S.l.], 2013. Disponível em: <<http://www.osirix-viewer.com/Roadmap.html>>. Acesso em: 18 set. 2013.
- PAIVA, Anselmo C.; SEIXAS, Roberto de B.; GATTAS, Marcelo. **Introdução à Visualização Volumétrica**. 1999. 16 p. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Departamento de Informática, PUC-Rio, Rio de Janeiro.
- VIENNA UNIVERSITY OF TECHNOLOGY. [S.l.], 2013. Disponível em: <<http://www.tuwien.ac.at/en/>>. Acesso em: 20 set. 2013.