UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISUALIZAÇÃO VOLUMÉTRICA DE IMAGENS DICOM PARA IOS

MARCELO DA MATA OLIVEIRA

BLUMENAU
2013

2013/1-23

MARCELO DA MATA OLIVEIRA

VISUALIZAÇÃO VOLUMÉTRICA DE IMAGENS DICOM PARA IOS

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

2013/1-23

VISUALIZAÇÃO VOLUMÉTRICA DE IMAGENS DICOM PARA IOS

Por

MARCELO DA MATA OLIVEIRA

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. Antonio Carlos Tavares, M. Sc. – FURB

Blumenau, dia 10 de julho de 2013

Dedico este trabalho a minha família, que sempre me apoiou nos meus estudos e nas minhas decisões para alcançar meus objetivos.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que mesmo longe, sempre esteve presente.

À minha noiva, pela paciência e compreensão durante todo este tempo de trabalho e estudo.

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

Aos meus amigos e companheiros do curso, que contribuíram durante todo o curso com novas idéias e conhecimento.

A todos os professores do curso, que sempre se colocaram disponíveis para conversar sobre as diferentes áreas de pesquisa da computação.

A mente que se abre a uma nova ideia jamais volta ao seu tamanho original.

Albert Einstein

RESUMO

Este trabalho apresenta a especificação e implementação de uma aplicação de visualização volumétrica de imagens DICOM para iOS. Uma biblioteca desenvolvida neste trabalho para ler o tipo de arquivo definido no padrão DICOM, 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. 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. O volume é visualizado nas três direções anatômicas usando a representação por `voxels` sem implementar os algoritmos de visualização científica, como *ray casting* ou *marching cubes*.

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

ABSTRACT

This work presents the specification and implementation of a volumetric visualization application of DICOM images to iOS. The library developed is responsible to read the file type defined in DICOM standard, reads the computerized medical examination files selected by the application's user and the images are extracted from each one of these files. Each image represents a slice. These slices are singly displayed in 2D visualization. In 3D visualization, they are organized in three dimension space to form the volume that represents the body part which the examination was taken. The volume is visualized in three anatomical directions using voxels representations without the implementation of scientific visualization algorithms, as ray casting or marching cubes.

Key-words: DICOM standard. Volumetric visualization. iOS.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ACR – *American College Radiology*

CenPRA – Centro de Pesquisas Renato Archer

CTI – Centro de Tecnologia da Informação Renato Archer

DICOM – *Digital Imaging and COmmunication in Medicine*

GDCM – *Grassroots Digital imaging and Communication in Medicine*

IDE – *Integrated Development Environment*

JPEG – *Join Photographic Experts Group*

MCT - Ministério da Ciência e Tecnologia

NEMA – *National Electrical Manufacturers Associations*

OpenGL ES – *Open Graphics Library for Embedded Systems*

URL - *Uniform Resource Locator*

UML - *Unified Modeling Language*

VR – *Value Representation*

VTK – *Visualization ToolKit*

SUMÁRIO

1 INTRODUÇÃO

Na sociedade moderna o uso de tecnologia computacional tornou-se indispensável em qualquer área. Hoje algumas facilidades de visualização de dados que estas tecnologias proporcionam, alavancam a eficiência e a produtividade do meio onde são aplicadas. Umas das áreas que se aproveitam de tais ferramentas é a medicina. Vários exames médicos são feitos de forma computadorizada, como a tomografia computadorizada e ressonância magnética.

De acordo com Monteiro (2005, p. iv), "Um dos principais avanços no diagnóstico médico é a utilização de métodos não invasivos para a obtenção de imagens de seções transversais do interior do corpo humano, forma de diagnóstico que tende a aumentar ano a ano".

O uso de dispositivos móveis pelos médicos facilita ainda mais a utilização de ferramentas para visualização de exames. Desta forma não se tem problemas em mover-se com o dispositivo de visualização, já que o médico pode deslocar-se com seu aparelho facilmente.

Visto acima, este trabalho apresenta o padrão *Digital Imaging and COmmunication in Medicine* (DICOM) e o desenvolvimento de uma biblioteca para ler o tipo de arquivo definido por este padrão. São também expostos os algoritmos para gerar a visualização em duas dimensões, a visualização volumétrica e o fatiamento do volume em três direções anatômicas. Com esta biblioteca e estes algoritmos, desenvolveu-se uma aplicação que permite que as imagens adquiridas dos arquivos DICOM de um exame médico possam ser analisadas no formato 3D, visualizando-se partes internas e externas delas, utilizando a aplicação em um dispositivo móvel com o sistema operacional iOS.

1.1 OBJETIVOS DO TRABALHO

O objetivo desse trabalho é desenvolver um aplicativo que permita realizar a visualização volumétrica de imagens no padrão DICOM na plataforma iOS.

Os objetivos específicos do trabalho são:

a) ler um arquivo no formato DICOM que se encontra no dispositivo móvel;
b) gerar a visualização 2D a partir de imagens adquiridas no arquivo DICOM;
c) gerar a visualização 3D a partir de imagens adquiridas no arquivo DICOM.

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 apresentas as conclusões e sugestões para trabalhos futuros que possam ser desenvolvidos a partir deste.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é apresentado o padrão DICOM. Posteriormente, na seção 2.2 são descritos os algoritmos de visualização volumétrica. Na seção 2.3 é abordada a plataforma iOS. Na seção 2.4 são descritos três trabalhos correlatos.

2.1 PADRÃO DICOM

O padrão DICOM iniciou seu desenvolvido por um comitê formado pelo American College Radiology (ACR) e National Electrical Manufacturers Assiciations (NEMA), em 1983. O objetivo era estabelecer um canal único para transferência de informações médicas, padronizando a comunicação, apresentação e armazenamento (RÚBIO, 2003, p. 27).

A primeira versão foi lançada em 1985. Posteriormente foram lançadas outras duas versões sendo a segunda versão em 1988 e a terceira em 1993, que é a mais recente. As alterações realizadas na última versão surgiram de problemas da primeira e da segunda versões, além disso foram criados novos processos, de modo especial o protocolo de comunicação para rede. O padrão atualmente está completo, mas pode sofrer alterações caso necessário (MONTEIRO, 2005, p. 54).

Segundo Franceschi (2006, p. 66), interfaces DICOM estão disponíveis para conexão de qualquer combinação de categorias de equipamentos de imagem digital. Como equipamentos de aquisição de imagens, programas para arquivamento de imagens, estações de trabalho para processamento e visualização de imagens e dispositivos de impressão. Desta forma, máquinas de diferentes fabricantes podem comunicar-se, desde que implementem o protocolo corretamente.

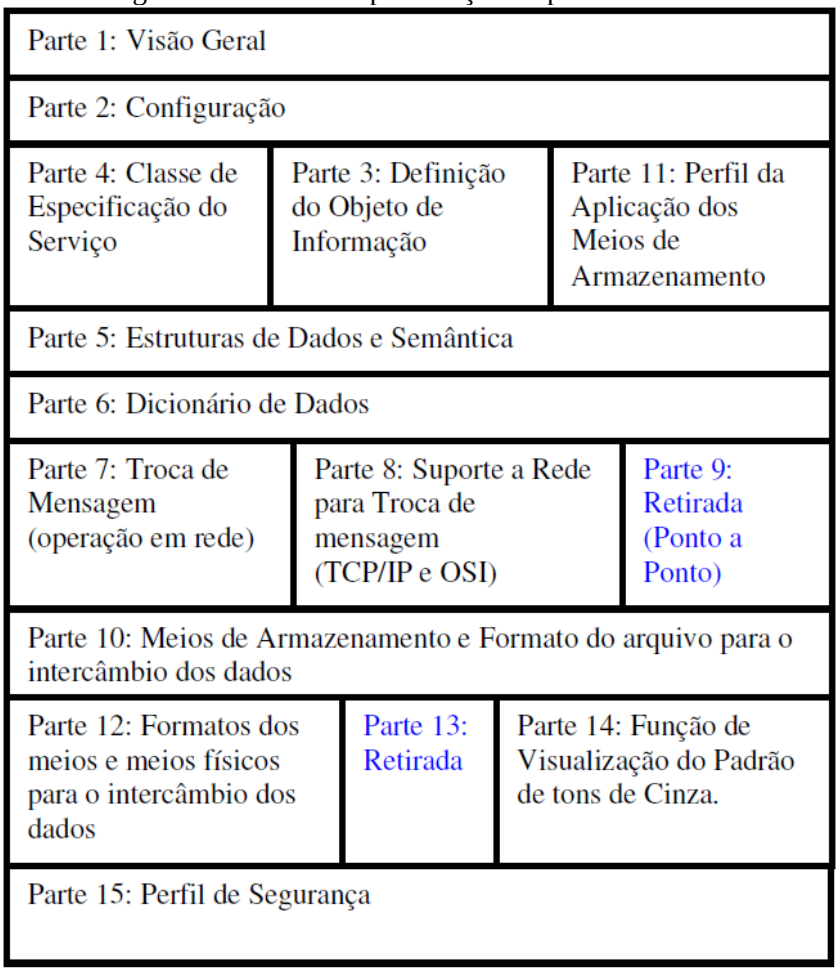
O padrão DICOM permite que informações dos pacientes e as imagens sejam armazenadas juntas. Há uma estrutura para manter estas informações, que é formada por `tags` que as delimitam. A imagem armazenada é baseada no formato *Join Photographic Experts Group* (JPEG), com ou sem compressão, por consequência do aparelho gerador. Cada imagem representa uma fatia do exame médico (MONTEIRO, 2005, p. 55).

A especificação do padrão DICOM é dividida em quinze partes, conforme pode ser observado na Figura 1. A parte dez é a que especifica os meios de armazenamento e o formato do arquivo para a comunicação dos dados.

O arquivo é formatado em um conjunto de dados, que por sua vez é formado pelos elementos de dados. Cada elemento de dados é constituído por uma `tag` (etiqueta), que possui o valor de um identificador, este valor é representado em hexadecimal. Após a `tag` há um campo que especifica o tipo e o formato do valor do elemento de dados, chamado *Value*

Representation (VR). Posteriormente há o campo com o tamanho do valor do elemento de dados. Por fim, há o campo valor (DOMETERCO, 2002, p. 20). A Figura 2 apresenta como é constituído o conjunto de dados.

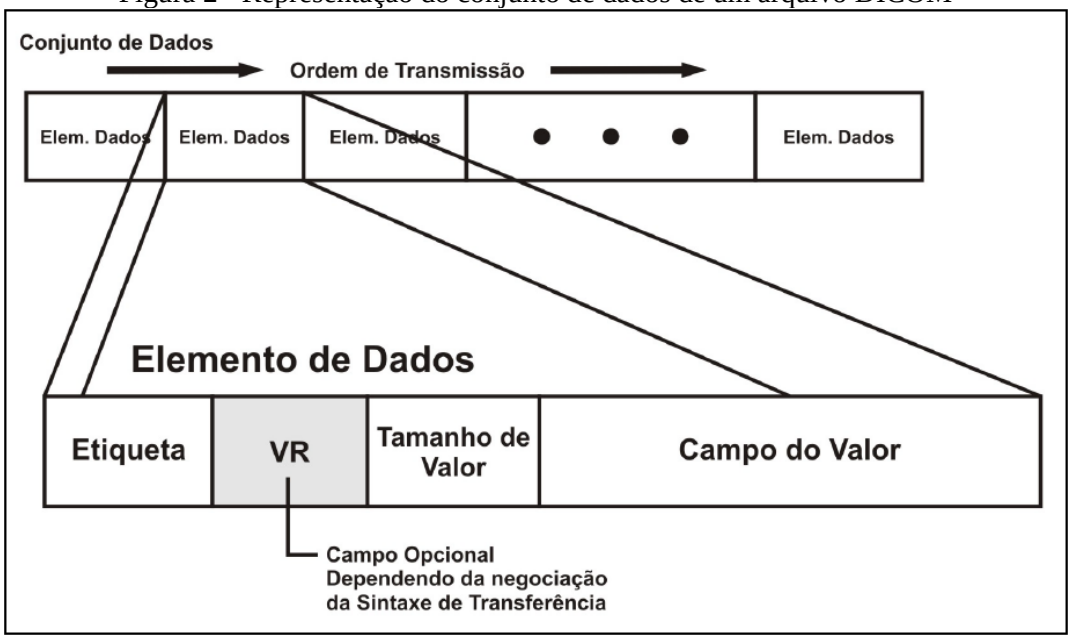
Figura 1 - Partes da especificação do padrão DICOM



Fonte: Monteiro (2005, p. 62).

Começaram a ser padronizadas, a partir de 1995, as principais modalidades de diagnóstico por imagem, como tomografia computadorizada e ressonância magnética (FRANCESCHI, 2006, p. 48). Nestas imagens cada bloco da estrutura em questão é representado por um *voxel*, definido como um elemento de volume. Outra interpretação do *voxel* é de um pequeno hexaedro definido em torno do valor amostrado. Cada um denota um (ou mais) valor(es) para uma pequena região espacial, não tendo sobreposição entre eles. O *voxel* pode representar diferentes tipos de dados e de grandeza como por exemplo densidade, pressão, temperatura, cor e opacidade (DOMETERCO, 2002, p. 23).

Figura 2 - Representação do conjunto de dados de um arquivo DICOM



Fonte: Roepke (2010, p. 17).

2.2 VISUALIZAÇÂO VOLUMÉTRICA

Facilitar o entendimento de um determinado problema, que gera muitos dados, partindo da representação visual dos dados é o principal objetivo da visualização científica. Desta forma a visualização volumétrica é utilizada para a exploração visual de dados volumétricos. Esta área passou a constituir uma área de pesquisa separada da computação gráfica (SILVA, 2003, p. 23).

Aplicações envolvendo imagens na medicina têm aspecto tridimensional. É necessário que estruturas internas ao volume de dados sejam visualizadas. A visualização volumétrica aplicada desta forma na medicina é feita utilizando imagens bidimensionais geradas por exames como tomografia computadorizada e ressonância magnética (FREITAS, 2002, p. 11).

Segundo Rúbio (2003, p. 3), há muitas perspectivas de investigação no espaço de visualização volumétrica. Vários trabalhos aperfeiçoaram os algoritmos já existentes, como a redução do tempo necessário para obter uma imagem, além de que vários algoritmos foram desenvolvidos.

A conversão de valores reais para a posterior visualização, como cor e opacidade, é um dos principais problemas na visualização volumétrica. Para realizar tal processo utilizam-se funções de transferências. Estas funções classificam os `voxels` que formam o volume, e por isso são importantes para o resultado e a qualidade das imagens visualizadas (PRAUCHNER, 2005, p. 14).

Conforme Elvins (1992 apud SEIXAS, 1997, p. 4), os métodos de visualização volumétrica são normalmente divididos em duas categorias, que são a visualização volumétrica direta, que não necessita de representação geométrica, e a visualização por extração de superfícies, que utiliza primitivas geométricas como triângulos ou polígonos mais gerais.

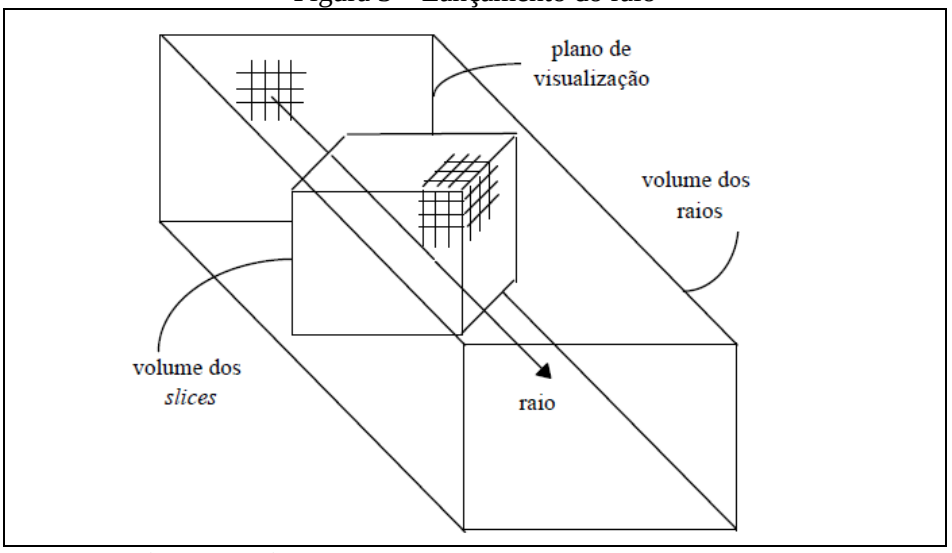
2.2.1 Visualização direta de volumes

Este método mapeia diretamente os `voxels` no espaço de visualização, sem uso de primitivas geométricas. É muito útil quando o objetivo é visualizar o interior de um objeto. Este método de visualização demanda grande quantidade de recursos computacionais, porém obtendo como produto imagens de qualidade muito boa (RÚBIO, 2003, p. 7).

Vários fatores podem influenciar a geração de uma imagem com a visualização direta de volumes, como a origem do volume de dados, os parâmetros utilizados durante o processo de visualização e o algoritmo de visualização. As alterações realizadas em um destes fatores, mesmo que mínima, pode mudar completamente a visualização dos dados. Os principais algoritmos que utilizam este método são *splatting*, *shear warp* e *ray casting* (SILVA, 2003, p. 11).

O algoritmo *ray casting* é o mais usado quando necessita-se de uma maior qualidade na visualização dos dados volumétricos (ELVINS, 1992 apud PAIVA; SEIXAS; GATTASS, 1999, p. 73). Este algoritmo consiste no lançamento de um raio a partir de cada `pixel` da imagem. A cor e a opacidade são acumuladas ao longo do raio até se determinar o valor final da cor e opacidade para o `pixel` (PAIVA; SEIXAS; GATTASS, 1999, p. 73). A Figura 3 demosntra o lançamento do raio.

Figura 3 – Lançamento do raio



Fonte: Paiva (1999, p. 74).

O *ray casting* pode ser facilmente paralelizado, pois o lançamento dos raios são independentes entre si. O algoritmo pode ser definido em uma forma geral conforme a Figura 4.

Figura 4 – *Ray casting*

```
InitRay( ) // inicializa  parâmetros de lançamento dos
raios
para  i de 1 a ImageHeight   // loop de lançamento
   para j de 1 a ImageWidth
       UpdateRay (ray, i, j )
       image[i,j] = RayCast(ray)
   fim para
fim para
```

Fonte: Paiva (1999, p. 74).

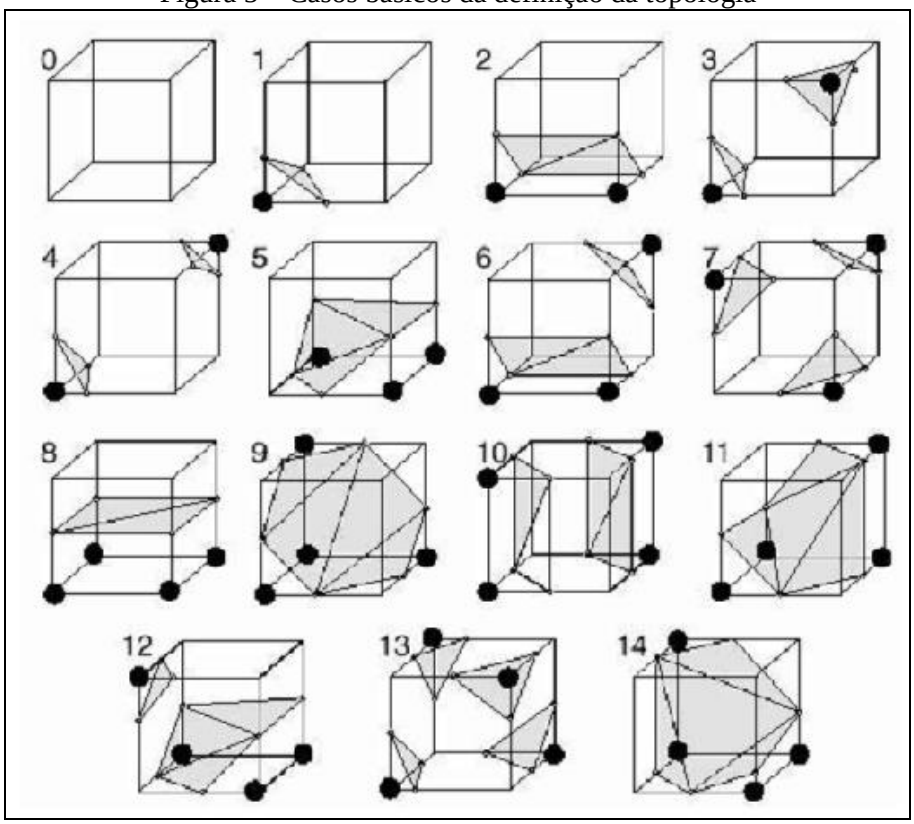
2.2.2 Visualização por extração de superfícies

Os algoritmos que beseiam-se neste método detalham a superfície em polígonos. No início do processo o usuário deve escolher um valor limiar, sendo que este valor será a base para a execução do algoritmo e para a obtenção de uma isosuperfície. Após esta etapa os métodos tradicionais de *rendering* de polígonos são utilizados para ter-se a visualização volumétrica da imagem (PAIVA; SEIXAS; GATTASS, 1999, p. 35).

As principais vantagens deste método de visualização são a velocidade e a pouca quantidade de armazenamento requerida. As desvantagens são a possibilidade de criação de superfícies onde elas não existem e a baixa qualidade em seu uso para visualizar algumas estruturas do corpo humano (MANSSOUR, 1998 apud RUBIO, 2003, p. 6). Dentre os principais algoritmos que utilizam este método estão o *marching cubes*, *dividing cubes* e *contour-connecting*.

No *marching cubes*, a superfície do volume é determinada pela verificação dos valores dos vértices dos `voxels` em relação ao da superfície. A partir desta verificação atribui-se 1 aos vértices com valor superior ao da superfície e 0 aos que possuem valor inferior. A partir desta codificação defini-se a topologia da superfície dentro dos `voxels`. Uma tabela determina a topologia da superfície dentro do `voxel` para cada caso possível (PAIVA; SEIXAS; GATTASS, 1999, p. 44). Através de considerações de rotação e complementaridade os casos da tabela são reduzidos de 256 para 15 (LORENSEN, 1987 apud PAIVA; SEIXAS; GATTASS, 1999, p. 45). A Figura 5 ilustra os 15 casos obtidos após a simplificação.

Figura 5 – Casos básicos da definição da topologia



Fonte: Freitas (2002, p. 20).

2.3 PLATAFORMA IOS

O sistema operacional iOS desenvolvido pela Apple é utilizado nos dispositivos móveis comercializados pela empresa como o iPad, iPhone e iPod *Touch*. A plataforma iOS foi construída com o conhecimento adquirido na criação do Mac OS X. Muitas das ferramentas e tecnologias utilizadas no desenvolvimento desta plataforma têm raízes neste sistema operacional também (APPLE INC, 2012).

O iOS SDK provê suporte para o desenvolvimento de aplicações para o iOS e inclui um conjunto completo de ferramentas, compiladores e *frameworks*. Com o *Integrated Development Environment* (IDE) Xcode pode-se desenvolver aplicações para executar no iPhone, iPad e iPod *Touch*. Para testes pode ser utilizado um simulador que acompanha a IDE que deve ser executado em um Mac OS X (APPLE INC, 2012).

O núcleo de controle do sistema operacional iOS é dividido em quatro camadas, como é demonstrado na Figura 6.

A camada Core OS contém características de nível mais baixo que as demais. Elas não são utilizadas diretamente nas aplicações, mas é a base de outras estruturas do sistema operacional e é desenvolvida na linguagem C. A camada Core Services por sua vez, contém serviços de sistemas fundamentais utilizados pelos aplicativos, também desenvolvida na

linguagem C. A camada seguinte, a camada Media, contém as tecnologias de gráfico, áudio e vídeo. É desenvolvida tanto na linguagem C quanto em Objective-C, sendo nesta camada que se inclui a tecnologia *Open Graphics Library for Embedded Systems* (OpenGL ES). No topo, está a camada Cocoa *Touch*, onde se implementa a interface dos aplicativos para o iOS, utilizando a linguagem Objective-C (APPLE INC, 2012).

Figura 6 - Camadas do núcleo de controle

Fonte: Apple Inc. (2012).

A API OpenGL ES fornece ferramentas para realizar desenhos em 2D e 3D. É uma API gratuita e multiplataforma utilizada em aparelhos como os celulares. Possui um conjunto de procedimentos e funções que possibilitam especificar os objetos e as operações utilizadas em imagens de alta qualidade gráfica (KHRONOS GROUP, 2012).

2.4 TRABALHOS CORRELATOS

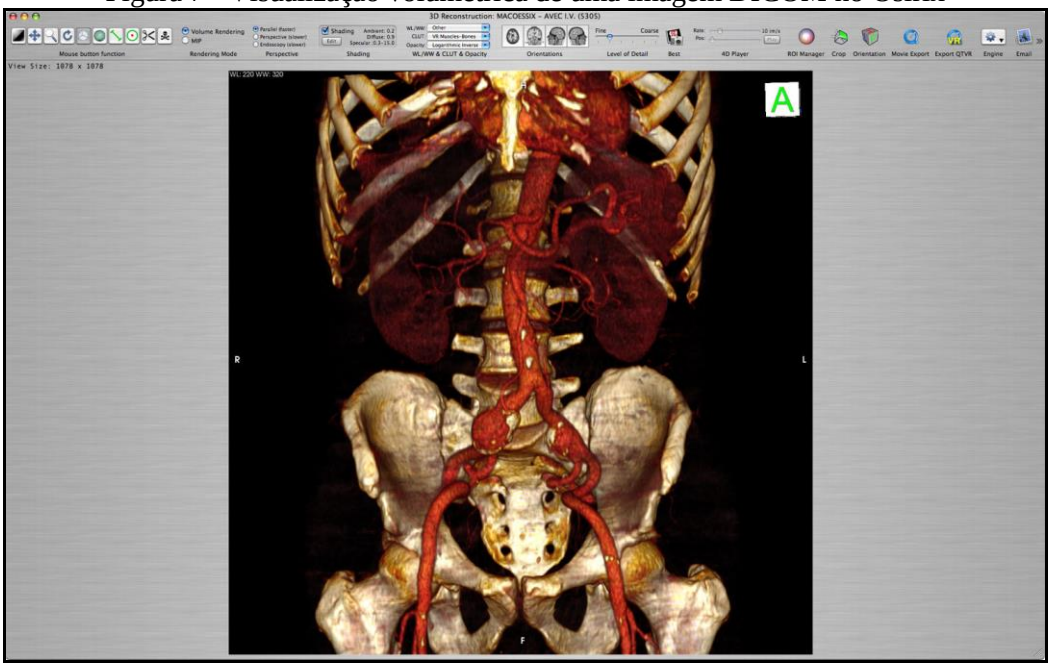
Foram selecionados três trabalhos que abordam o assunto de visualização volumétrica para computadores pessoais e dispositivos móveis na plataforma iOS: Osirix (2012), InVesalius (2012) e "Visualizador de Imagens Radiológicas 2D para Iphone" (ROEPKE, 2010).

2.4.1 Osirix

O Osirix é um software de processamento de imagens dedicado a imagens DICOM. É totalmente compatível com o padrão DICOM. É considerada a mais completa ferramenta de visualização de imagens DICOM, sendo desenvolvido em Objective-C e tendo seu código como *opensource* somente para a plataforma MAC OS. Primeiramente a ferramenta foi desenvolvida para MAC OS, sendo posteriormente adaptada em uma versão paga para o iOS (OSIRIX, 2012). Esta ferramenta é resultado de mais de cinco anos de pesquisa e desenvolvimento em imagem digital.

A ferramenta suporta uma arquitetura de *plug-ins* completa, podendo assim ser customizada. Ela foi projetada para navegação e visualização de imagens multimodalidade e multidimensionais. É possível realizar a visualização em 2D e 3D. Em alguns tipos de imagens médicas, como imagens cardíacas é possível ainda obter uma visualização 4D e 5D. O visualizador 3D oferece todos os modos de visualização modernos (OSIRIX, 2012). A Figura 7 mostra o visualizador em uma visualização de volumes.

Figura 7 - Visualização volumétrica de uma imagem DICOM no Osirix



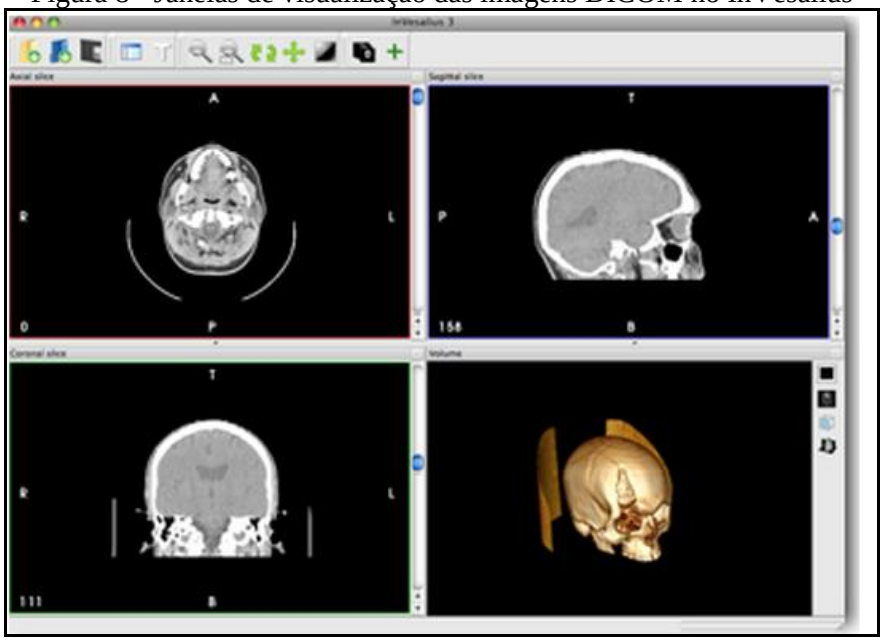
Fonte: Osirix (2012).

2.4.2 InVesalius

O InVesalius é um software que tem o objetivo de auxiliar o diagnóstico e o planejamento de cirurgias. O InVesalius foi desenvolvido nas linguagens de programação C++ e Python, utilizando bibliotecas como *Visualization ToolKit* (VTK) e *Grassroots Digital imaging and Communication in Medicine* (GDCM). É um software livre aplicado na área da saúde. O programa foi desenvolvido pelo antigo CenPRA, atual Centro de Tecnologia da Informação Renato Archer (CTI), unidade do Ministério da Ciência e Tecnologia (MCT). O software adquire imagens 2D, provenientes de equipamentos como tomografia computadorizada ou ressonância magnética e cria modelos 3D correspondentes às estruturas anatômicas. O aplicativo é versátil, tendo contribuído com diversas áreas na medicina, odontologia, veterinária, arqueologia e engenharia (INVESALIUS, 2012).

Na Figura 8 observa-se a tela de visualização subdividida em quatro janelas, sendo uma delas a de visualização de volumes.

Figura 8 - Janelas de visualização das imagens DICOM no InVesalius



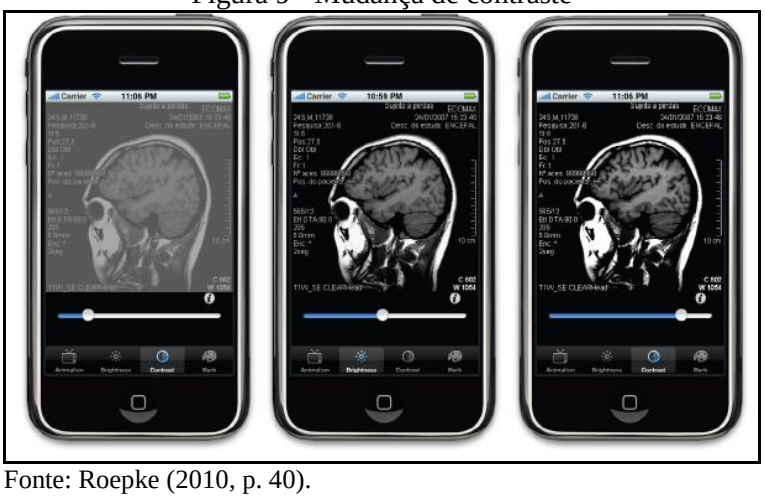
Fonte: Osirix (2012).

2.4.3 Visualizador de imagens radiológicas 2D para iphone

No trabalho de Roepke (2010) o objetivo foi construir uma ferramenta para visualizar imagens radiológicas no formato JPEG para o iPhone. Seu trabalho foi desenvolvido utilizando a linguagem Objective-C. Foram utilizadas no desenvolvimento os *frameworks* UIKit, *Foundation*, *Quartz*, *Core* e *Core Graphics*, que são fornecidos pela própria Apple.

As imagens no formato JPEG são adquiridas por meio de uma *Uniform Resource Locator* (URL), o que permite visualizá-las de um servidor. O aplicativo desenvolvido permite também aumentar ou diminuir brilho e contraste, além de possibilitar marcar regiões da imagem e enviá-la por email (ROEPKE, 2010). Na Figura 9 pode verificar-se a mudança de contraste realizada em uma imagem DICOM.

Figura 9 - Mudança de contraste



Fonte: Roepke (2010, p. 40).

2.4.4 Características dos trabalhos correlatos

Com base nos trabalhos correlatos, observa-se as principais características dos mesmos. No Quadro 1 apresenta-se uma comparação entre os trabalhos observando as características de cada um.

Quadro 1 - Comparativo entre os trabalhos correlatos

Características\Trabalhos correlatos	InVesalius	Osirix	Roepke (2010)
Ler arquivos DICOM	X	X	
Visualização em duas dimensões	X	X	X
Tratamento das imagens	X	X	X
Visualização volumétrica	X	X	
Visualização interna do volume	X	X	
Visualização em mais de três dimensões		X	
Separação do volume em componentes	X	X	
Fatiamento do volume	X	X	
Geração da imagem em outras direções anatômicas	X	X	
Aplicação para dispositivos móveis		X	X

3 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo são expostas as etapas do desenvolvimento da aplicação de visualização de imagens DICOM e da biblioteca utilizada para recuperar imagens dos arquivos `dcm`. 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 visualização volumétrica e em duas dimensões de imagens DICOM possui o seguintes requisitos:

a) ler o cabeçalho e as imagens de um arquivo DICOM (Requisito Funcional – RF);
b) apresentar a sequência de imagens em formato 2D contidas no arquivo DICOM em na direção anatômica que a imagem foi capturada (RF);
c) realizar a renderização volumétrica das imagens DICOM (RF);
d) realizar o fatiamento do volume em três direções anatômicas, axial, sagital e coronal (RF);
e) ser implementado utilizando a linguagem de programação Objective-C (Requisito Não-Funcional – RNF);
f) ser implementado utilizando o ambiente de desenvolvimento XCode 4 (RNF);
g) ser desenvolvido para executar em dispositivos móveis como o iPhone, iPad e iPod Touch (RNF).

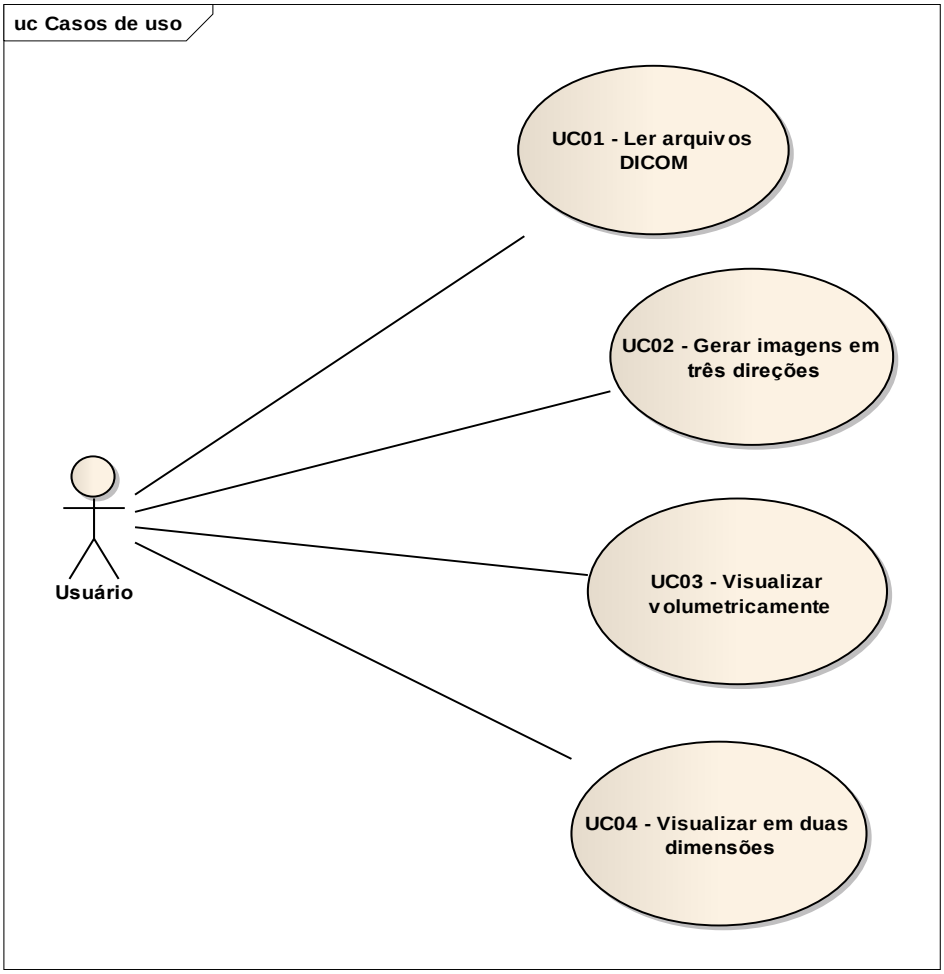
3.2 ESPECIFICAÇÃO

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). A ferramenta Enterprise Architect foi utilizada na especificação. 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 10 pode-se observar o diagrama de casos de uso com o ator e os casos de uso.

Figura 10 - 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 2 descreve em detalhes este caso de uso.

Quadro 2 - Caso de uso ler arquivos DICOM

Ler arquivos DICOM	
Pré-condições	Estar com a tela para selecionar um exame aberta.
Cenário Principal	1) O usuário seleciona um exame. 2) A aplicação obtém o diretório do exame. 3) A aplicação obtém todos os arquivos DICOM dentro do diretório . 4) A aplicação lê todos os arquivos. 5) A aplicação obtém as imagens de todos os arquivos.
Fluxo Alternativo	1) O usuário seleciona o botão `cancel`. 2) É apresentada a tela de visualização volumétrica sem objetos. 3) O usuário seleciona o botão no canto superior direito da tela. 4) Abre-se a tela para a seleção de exames. 5) O usuário seleciona um exame. 6) A aplicação obtém o diretório do exame. 7) A aplicação obtém todos os arquivos DICOM dentro do diretório . 8) A aplicação lê todos os arquivos. 9) A aplicação obtém as imagens de todos os arquivos.
Pós-condições	A tela de seleção de exame é fechada.

3.2.1.2 Gerar imagens em três direções

Este caso de uso mostra como é a relação entre o usuário e a geração de imagens em três direções. Para ver o detalhe da descrição deste caso de uso consulte o Quadro 3.

Quadro 3 - Caso de uso Gerar imagens em três direções

Gerar imagens em três direções	
Pré-condições	O usuário ter selecionado um exame do qual as imagens foram obtidas na direção axial.
Cenário Principal	1) A imagem é visualizada volumetricamente na direção axial. 2) O usuário solicita que o volume seja visualizada na direção sagital. 3) A aplicação mostra o volume com as imagens na direção sagital. 4) O usuário solicita que volume seja visualizada na direção coronal. 5) A aplicação mostra o volume na direção coronal. 6) O usuário seleciona o botão `Rotate`. 7) A aplicação mostra o volume na direção coronal.
Fluxo Alternativo 1	1) A imagem é visualizada volumetricamente na direção axial. 2) O usuário solicita o fatiamento da imagem. 3) A aplicação visualiza o volume na direção axial. 4) O usuário toca a tela e desliza o dedo sobre a tela na vertical. 5) O fatiamento na direção axial é realizado. 6) O usuário seleciona o botão `Rotate`. 7) A aplicação mostra o volume na direção axial com as fatias visualizadas após o fatiamento.
Fluxo Alternativo 2	1) O usuário solicita o fatiamento da imagem na direção coronal. 2) A aplicação visualiza o volume na direção coronal. 3) O usuário toca a tela e desliza o dedo sobre a tela na horizontal. 4) O fatiamento na direção coronal é realizado. 5) O usuário seleciona o botão `Rotate`. 6) A aplicação mostra o volume na direção coronal com as fatias visualizadas após o fatiamento.
Fluxo Alternativo 3	1) O usuário solicita o fatiamento da imagem na direção sagital. 2) A aplicação visualiza o volume na direção sagital. 3) O usuário toca a tela e desliza o dedo sobre a tela na horizontal. 4) O fatiamento na direção sagital é realizado. 5) O usuário seleciona o botão `Rotate`. 6) A aplicação mostra o volume na direção sagital com as fatias visualizadas após o fatiamento.
Pós-condições	A função de fatiamento não está habilitada.

3.2.1.3 Visualizar volumetricamente

Este caso de uso mostra como é a relação entre o usuário e a visualização volumétrica de imagens DICOM. Para ver o detalhe da descrição deste caso de uso consulte o Quadro 4.

Quadro 4 - Visualizar volumetricamente

Visualizar volumetricamente	
Pré-condições	O usuário ter selecionado um exame com arquivos DICOM.
Cenário Principal	1) A aplicação visualiza volumetricamente o exame. 2) O usuário solicita a rotação do volume. 3) A aplicação rotaciona o objeto em torno do seu eixo. 4) O usuário solicita o fatiamento da imagem na direção sagital. 5) A aplicação visualiza o volume na direção sagital com o volume rotacionado. 6) O usuário toca a tela e desliza o dedo sobre a tela na horizontal. 7) O fatiamento na direção sagital é realizado. 8) O usuário seleciona o botão `Rotate`. 9) A aplicação mostra o volume na direção sagital com as fatias visualizadas após o fatiamento. 10) O usuário solicita a rotação do volume. 11) A aplicação rotaciona o objeto em torno do seu eixo. 12) O usuário seleciona o botão superior direito.
Fluxo Alternativo 1	1) A aplicação visualiza volumetricamente o exame. 2) O usuário solicita a rotação da câmera. 3) A aplicação rotaciona a câmera. 4) O usuário solicita o fatiamento da imagem na direção sagital. 5) A aplicação visualiza o volume na direção sagital com o volume rotacionado. 6) O usuário toca a tela e desliza o dedo sobre a tela na horizontal. 7) O fatiamento na direção sagital é realizado. 8) O usuário seleciona o botão `Rotate`. 9) A aplicação mostra o volume na direção sagital com as fatias visualizadas após o fatiamento. 10) O usuário solicita a rotação da câmera. 11) A aplicação rotaciona a câmera. 12) O usuário seleciona o botão superior direito.
Pós-condições	A tela de seleção de exames é aberta.

3.2.1.4 Visualizar em duas dimensões

Este caso de uso mostra como é a relação entre o usuário e a visualização em duas dimensões de imagens DICOM. Para ver o detalhe da descrição deste caso de uso consulte o Quadro 5.

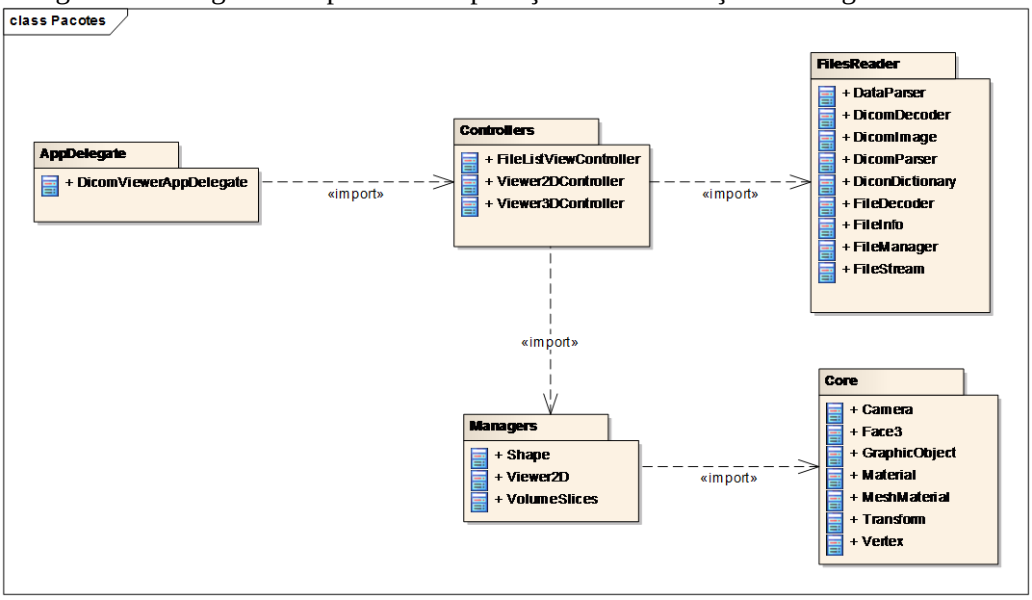
Quadro 5 - Visualizar em duas dimensões

Visualizar em duas dimensões	
Pré-condições	Estar com a tela para selecionar um exame aberta com a opção de selecionar o exame do crânio e o exame do joelho.
Cenário Principal	1) O usuário solicita visualizar o exame do crânio. 2) A tela de visualização 3D é aberta. 3) O usuário solicita visualizar 2D. 4) A tela de visualização 2D é aberta com uma imagem do exame do crânio. 5) O usuário altera o componente `slider` da tela. 6) A aplicação mostra a imagem referente a posição atual do `slider`. 7) O usuário seleciona o botão `cancel`.
Fluxo Alternativo 1	1) O usuário solicita visualizar o exame do joelho. 2) A tela de visualização 3D é aberta. 3) O usuário solicita o fatiamento na direção axial. 4) O volume é formado pelas imagens geradas na direção axial. 5) O usuário solicita a visualização 2D. 6) A tela de visualização 2D é aberta com uma imagem do exame do joelho. 7) O usuário altera o componente `slider` da tela. 8) A aplicação mostra a imagem referente a posição atual do `slider`. 9) O usuário seleciona o botão `cancel`.
Pós-condições	A tela de visualização 3D 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. Para facilitar o entendimento da organização geral das classes, na Figura 11 são ilustrados os pacotes e as classes que os compõem. Para a visualização volumétrica adaptou-se a biblioteca desenvolvida por Imianowsky (2013). O pacote `Controllers` possui classes adaptadas e o pacote `Core` além de classes adaptadas possui classes utilizadas de forma integral. Posteriormente cada pacote é descrito.

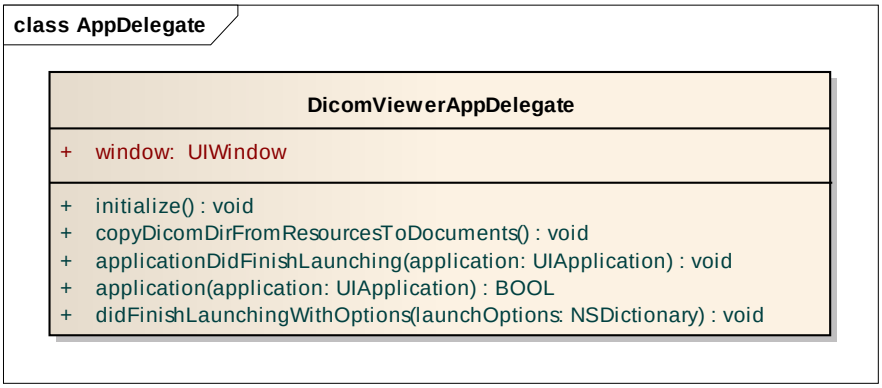
Figura 11 - Diagrama de pacotes da aplicação de visualização de imagens DICOM



3.2.2.1 Pacote `AppDelegate`

O pacote `AppDelegate` possui somente a classe `DicomViewerAppDelegate`. Esta classe é iniciada quando a aplicação inicia a execução. Ela é responsável por organizar os arquivos dos exames DICOM já inseridos na aplicação. Desta forma, posteriormente o usuário da aplicação poderá escolher qual desses exames visualizar. Veja a ilustração da classe `DicomViewerAppDelegate` na Figura 12. Ao término de sua execução é passada a execução para a classe `Viewer3DController` dentro do pacote `Controllers`.

Figura 12 - Diagrama de classes do pacote `AppDegate`



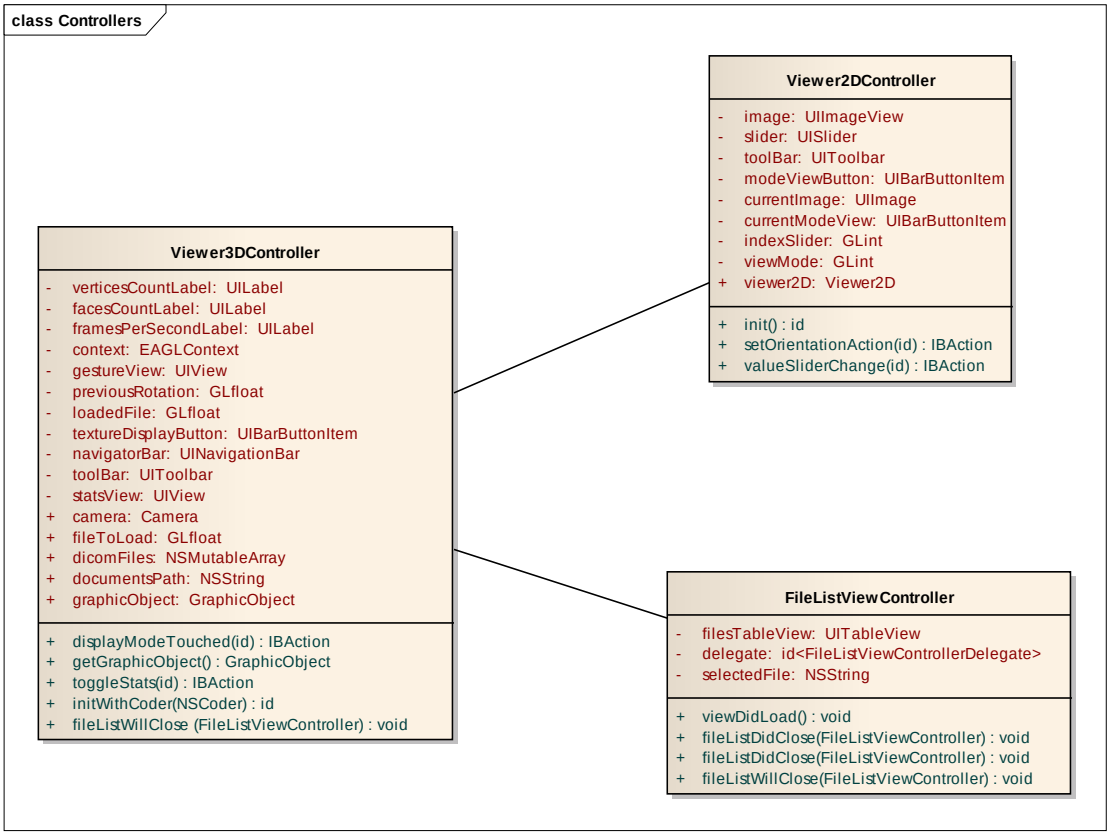
3.2.2.2 Pacote `Controllers`

Neste pacote encontram-se as classes responsáveis pela comunicação com camada de visão da aplicação. A classe `Viewer3DController` é a principal classe desse pacote. A tela principal da aplicação é a de visualização volumétrica, da qual esta classe é responsável pelo gerenciamento. Dessa forma, a partir desta classe que serão acessadas as outras classes deste pacote. Ainda nessa classe são recebidos os eventos para o fatiamento do volume gerado pelo exame em três direções anatômicas: sagital, axial e coronal.

No ínicio de sua execução, a classe `Viewer3DController`, instancia a classe `FileListViewController`, que por sua vez carrega todos os diretórios que possuem arquivos DICOM. Ao encontrar os arquivos a execução é retornada para o método `fileListWillClose` da classe `Viewer3DController` que irá acessar o pacote `FilesReader` para ler os arquivos `dcm` do exame.

Ao solicitar para ir para o modo de visualização em duas dimensões, a classe `Viewer3DController` inicia a execução da classe `Viewer2DController`, passando o exame a ser visualizado. A ilustração das classes desse pacote é apresentada na Figura 13.

Figura 13 - Diagrama de classes do pacote `Controllers`



Duas classes adaptadas da biblioteca desenvolvida por Imianowsky (2013) estão neste pacote. A classe `Viewer3DController` e `FileListViewController`. Na primeira as funcionalidades utilizadas foram a captura dos eventos gerados pelo toque na tela e a comunicação com a classe `FileListViewController`. Na segunda foi adaptado o método `viewDidLoad`.

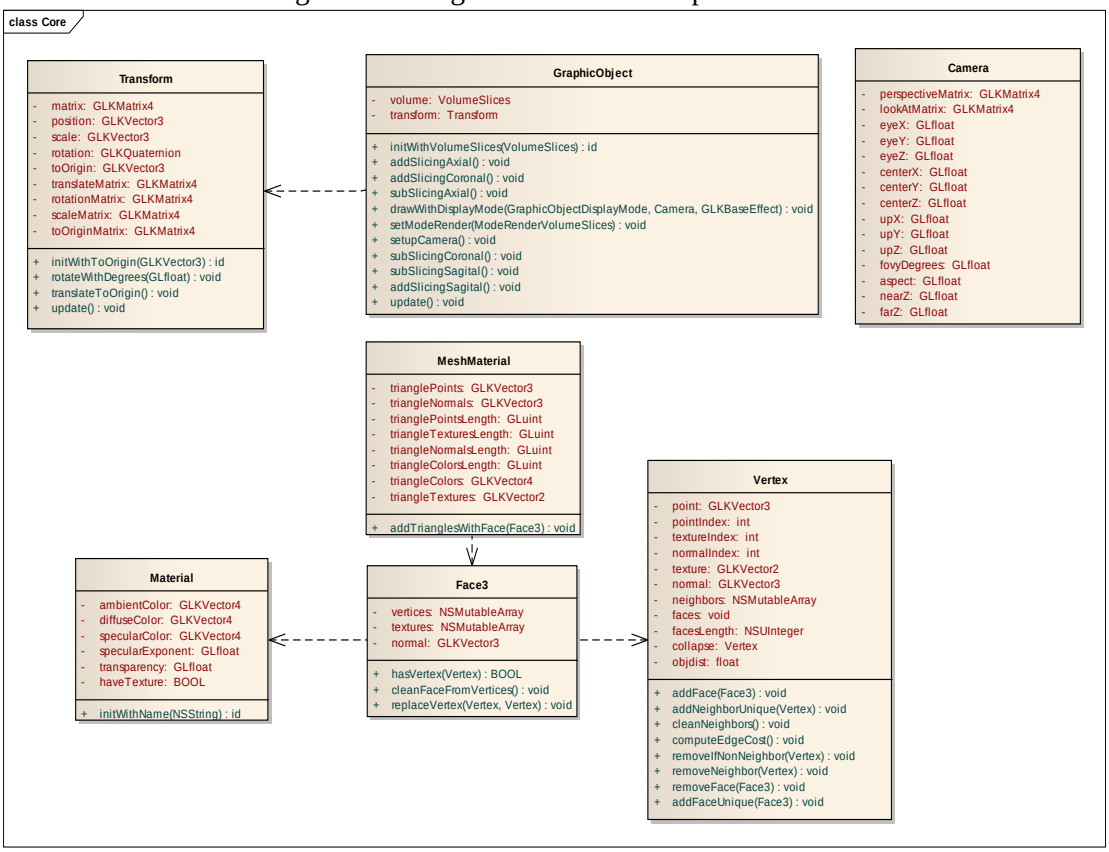
3.2.2.3 Pacote `Core`

Classes que trabalham com a modelagem geométrica da visualização volumétrica se encontram neste pacote. A classe `GraphicObject` é acessada pela `Viewer3DController` do pacote `Controllers`. No método `drawWithDisplayMode` está o algoritmo que desenha o volume na tela conforme os dados geométricos recebidos. A Figura 14 ilustra este pacote.

A classe `Transform` é utilizada pela classe `GraphicObject`. Ela é acessada para realizar a transformação geométrica dos dados. Conforme o volume vai sendo movimentado pelo o usuário os valores de suas matrizes vão sendo atualizados. Os valores para o posicionamento da câmera estão na classe `Camera`. Esta classe é acessada pela `Viewer3DController` do pacote `Controllers`, que recebe comando para rotacionar a câmera. Nas classes `MeshMaterial`, `Material`, `Face3` e `Vertex` os dados do volume são

organizados. Os dados geométricos gerados após a leitura das imagens DICOM são passados para estas classes e dessa forma são obtidas as faces que formam o volume.

Figura 14 - Diagrama de classes do pacote `Core`



Neste pacote existem classes adaptadas e utilizadas de forma integral da biblioteca desenvolvida por Imianowsky (2013). As que foram utilizada de forma integral são as classes `Camera`, `Transform`, `Material`, `Face3` e `Vertex`. As outras, `MeshMaterial` e `GraphicObject` foram adaptadas para serem utilizadas neste trabalho.

3.2.2.4 Pacote `FilesReader`

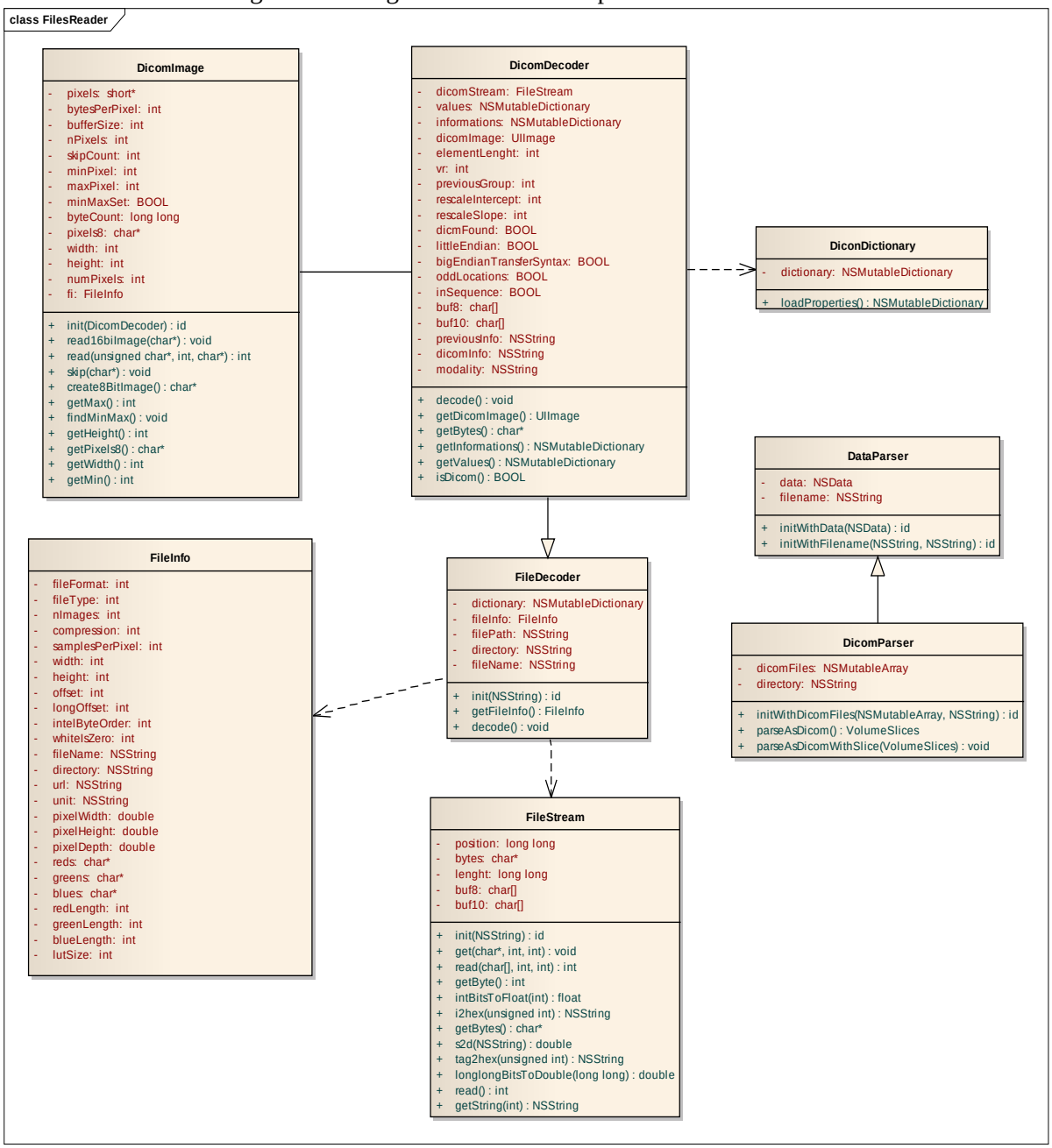
Este pacote é responsável por ler as imagens dos arquivos DICOM do diretório do exame selecionado. Observe a Figura 15 para visualizar o diagrama de classes deste pacote. A classe `DicomParser` é acessada pela `Viewer3DController` do pacote `Controllers`, e tem a função de obter o diretório do exame selecionado. No método `parseAsDicom` da classe `DicomParser` inicia a execução principal desse pacote. Nele é criada uma instância da classe `VolumeSlices` do pacote `Renderers` e `DicomDecoder`.

A classe `DicomDecoder` é acessada para ler cada arquivo `dcm` do diretório e recuperar a imagem em cada um destes. O método `decode` é o principal, nele é iniciada a leitura do arquivo e analizadas todas as `tags` pertinentes a posterior geração do volume. Ao localizar o

início de onde estão os dados da imagem, a classe `DicomImage` é acionada. Após este ponto são obtidas as características da imagem, como a quantidade de *bits* por `pixel`, o tamanho da imagem e os valores dos `pixels`.

As informações obtidas de cada arquivo são passadas para a classe `FileInfo`. A classe `FileStream` é responsável por obter cada *byte* do arquivo, ela é acessada pela classe `DicomDecoder`. Seus métodos têm a função de conseguir diferentes tipos de dados conforme a decrição da `tag` lida. Por fim, a classe `DicomDictionary` obtém o mapeamento entre valores e descrição de cada `tag`.

Figura 15 - Diagrama de classes do pacote `FilesReader`

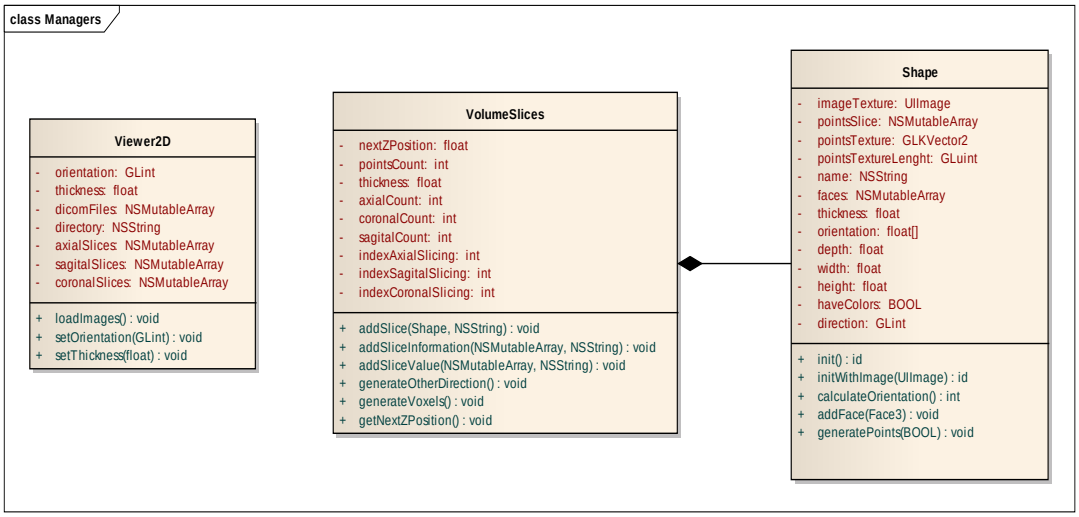


3.2.2.5 Pacote `Managers`

As principais classes deste pacote são a `VolumeSlices` e `Viewer2D`. A primeira desempenha um importante papel na organização dos dados para a visualização volumétrica. A segunda é relevante na visualização em duas dimensões. A classe `Shape` extende `MashMaterial` do pacote `Core`.

A classe `VolumeSlices` concentra o acesso a todos os dados utilizados na visualização volumétrica. Ela possui todas as fatias do exame. Um equipamento médico gera as imagens DICOM de um diretório de um exame em uma direção somente, que pode ser a axial, coronal ou sagital. Podem existir mais de um diretório em um exame podendo cada um destes terem sido obtidos em uma direção diferente. Dessa forma, nesta classe são geradas as fatias nas duas direções que faltam. Cada fatia é modelada pela classe `Shape`. Nos atributos desta classe observa-se que neles são armazenados valores obtidos de cada arquivo no pacote `FilesReader`. Ainda na classe `Shape` destaca-se o método `generatePoints` que irá gerar os pontos que serão utilizados na visualização volumétrica posteriormente na classe `GraphicObject` do pacote `Core`. O diagrama de classes deste pacote pode ser visto na Figura 16.

Figura 16 - Diagrama de classes do pacote `Managers`



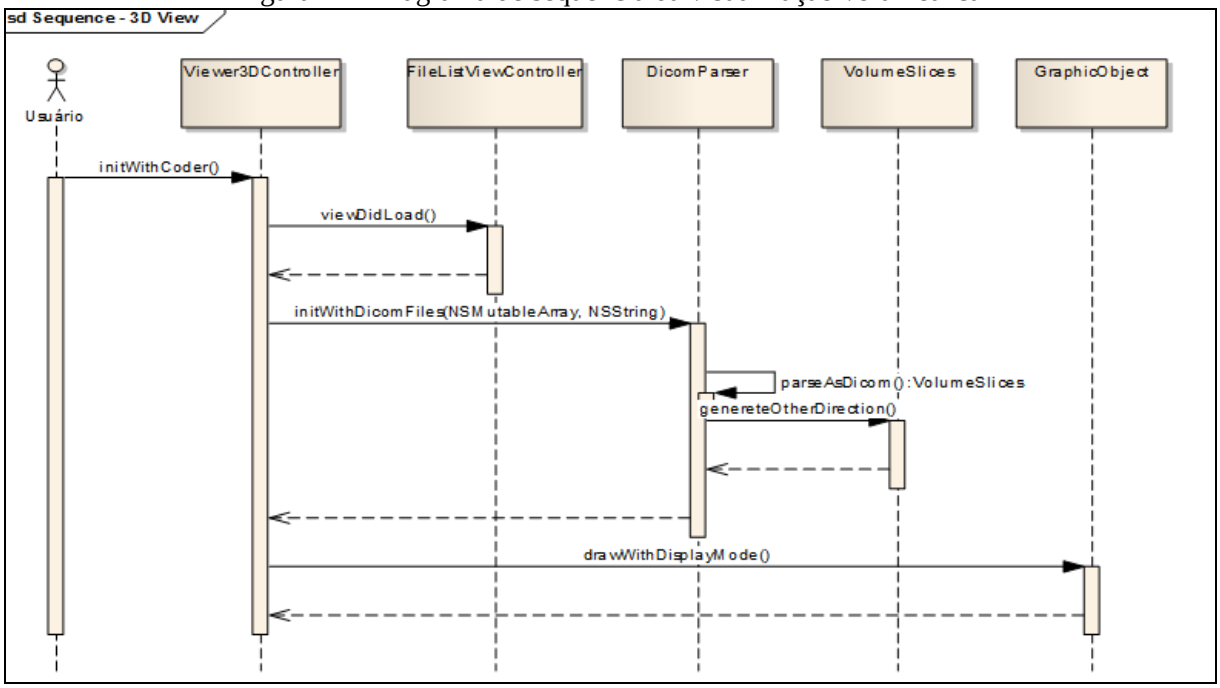
3.2.3 Diagrama de sequência

Esta seção apresenta o diagrama de sequência. Demonstra-se como ocorre a interação do usuário a aplicação. A Figura 17 ilustra este diagrama. É demontrada a interação entre o usuário e a aplicação quando vai realizar-se uma visualização volumétrica.

Nesta interação, é iniciada a aplicação no objeto `Viewer3DController`. O exame é escolhido na classe `FileListViewerController` e a execução retorna para o objeto anterior.

Este irá chamar o `DicomParser` para ler os arquivos e obter o volume encapsulado em `VolumeSlices`. Posteriormente o objeto `GraphicObject` desenha o volume na tela.

Figura 17 - Diagrama de sequência da visualização volumétrica



3.3 IMPLEMENTAÇÃO

A seguir são apresentadas 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 de programação utilizada no desenvolvimento de toda aplicação foi Objective-C. O ambiente de desenvolvimento utilizado foi o Xcode 4.5.1. As bibliotecas utilizadas nesta etapa são em sua maioria disponibilizadas pela Apple, são as seguintes: UIKit; Foundation; GLKit; CoreGraphics; OpenGLES.

A biblioteca UIKit foi utilizada na criação da interface com o usuário. A biblioteca Foundation, por sua vez, possui várias classes bases, como implementações de estruturas de dados e manipulação de arquivos. Classes utilizadas na modelagem geométrica do volume se encontram na biblioteca GLKit. Encontram-se na biblioteca CoreGraphics classes utilizadas na manipulação das imagens. Por fim, a biblioteca OpenGLES possui as classes utilizadas na visualização volumétrica.

A biblioteca desenvolvida por Imianowsky (2013) foi adaptada a este trabalho, sendo adaptadas classes com funções na visualização volumétrica. Além desta, os algoritmos

utilizados para a leitura dos arquivos DICOM foram adaptados da ferramenta de visualização e manipulação de imagens ImageJ (2013).

3.3.2 Aplicação de visualização de imagens DICOM

Primeiramente são lidos todos os arquivos `dcm` recuperando-se as imagens. Posteriormente é apresentado o volume gerado, que é formado pela organização de objetos nos quais são aplicadas como textura as imagens recuperadas.

A execução principal da aplicação inicia na classe `Viewer3DController`. No início do processamento, a classe `FileListViewController` é acionada. Ao usuário selecionar o exame a ser visualizado que está dentro da aplicação. Há um diretório com os vários arquivos `dcm` que são lidos para serem apresentados pela a aplicação, tanto volumetricamente quanto as imagens em duas dimensões.

3.3.2.1 Leitura dos arquivos DICOM

Nesta etapa são lidos todos os arquivos do diretório do exame selecionado. A classe `DicomParser` é instanciada logo após a seleção do exame médico a ser visualizado. Para cada arquivo é verificado se é um arquivo `dcm` ou não. No cabeçalho do arquivo, os quatro bytes posteriores ao de número 128, representam as letras D, I, C e M, como ocorre na linha 58. Desta forma, o método `isDicom` da classe `DicomDecoder` verifica se o arquivo corrente é DICOM ou não. Esta verificação é necessária pois também existem arquivos temporários utilizados para controle da aplicação no diretório. O código deste método é apresentado no Quadro 6.

Quadro 6 - Código do método `isDicom`

```
48.   -(BOOL)isDicom {
49.       if([[NSFileManager defaultManager] fileExistsAtPath:self.directory]) {
50.           char *buffer = malloc(DICOM_SIZE_HEAD * (sizeof(unsigned char)));
51.           [dicomStream get:buffer:0 :DICOM_SIZE_HEAD];
52.
53.           char b0 = buffer[128] & 255;
54.           char b1 = buffer[129] & 255;
55.           char b2 = buffer[130] & 255;
56.           char b3 = buffer[131] & 255;
57.
58.           if(b0 == 'D' && b1 == 'I' && b2 == 'C' && b3 == 'M') {
59.               return true;
60.           }
61.       }
62.
63.       return false;
64.   }
```

O método `decode` da classe `DicomDecoder` é o principal da leitura do arquivo DICOM. Nele cada `tag` do arquivo é lida, verificando após a leitura a relevância ou não do dado armazenado para a aplicação. Um trecho do código, onde é verificado se na `tag` corrente estão

os dados dos `pixels`, é ilustrado no Quadro 7. Entre a linha 232 e 241 está a verificação da `tag` que indica as informações dos `pixels`.

Quadro 7 - Trecho do código do método `decode`

```
141.     if(tag == TRANSFER_SYNTAX_UID) {
142.         s = [dicomStream getString:elementLenght];
143.          [self addInfo:tag :s];
144.
145.         if ([self indexOf:s:@"1.2.840.10008.1.2.2"]>=0) {
146.             bigEndianTransferSyntax = true;
147.          }
148.     }
                                                . . .
204.     else if(tag == WINDOW_WIDTH) {
205.         width = [dicomStream getString:elementLenght];
206.         index = [self indexOf:width :@"\\"];
207.         if (index!=1) {
208.             [width substringFromIndex:index+1];
209.         }
210.         windowWidth = [dicomStream s2d:width];
211.         [self addInfo:tag :width];
212.     } else if(tag == RESCALE_INTERCEPT) {
213.         intercept = [dicomStream getString:elementLenght];
214.         rescaleIntercept = [dicomStream s2d:intercept];
215.         [self addInfo:tag :intercept];
216.     } else if(tag == RESCALE_SLOPE) {
217.         slop = [dicomStream getString:elementLenght];
218.         rescaleSlope = [dicomStream s2d:slop];
219.         [self addInfo:tag :slop];
220.     } else if(tag == RED_PALETTE) {
221.         [self.fileInfo setReds: [self getLut:elementLenght]];
222.         [self.fileInfo setRedLength: elementLenght];
223.         [self addInfoInt:tag :elementLenght/2];
224.     } else if(tag == GREEN_PALETTE) {
225.         [self.fileInfo setGreens: [self getLut:elementLenght]];
226.         [self.fileInfo setGreenLength: elementLenght];
227.         [self addInfoInt:tag :elementLenght/2];
228.     } else if(tag == BLUE_PALETTE) {
229.         [self.fileInfo setBlues: [self getLut:elementLenght]];
230.         [self.fileInfo setBlueLength: elementLenght];
231.         [self addInfoInt:tag :elementLenght/2];
232.     } else if(tag == PIXEL_DATA) {
233.         //Inicio dos dados da imagem
234.         if (elementLenght!=0) {
235.             [self.fileInfo setOffset: [dicomStream getPosition]];
236.             [self addInfoInt:tag :[dicomStream getPosition]];
237.             decodingTags = false;
238.         } else {
239.             [self addInfo:tag :nil];
240.         }
241.   }
```

3.3.2.2 Visualização volumétrica

O processamento feito para realizar a visualização volumétrica inicia quando é concluída a leitura das imagens. A classe `DicomParser` possui o método `parseAsDicom`, que após cada imagem lida cria um objeto da classe `Shape`, que representa a abstração das fatias do volume. Os valores importantes como a distância entre fatias, tamanho da imagem e direção são recuperados da lista com os dados das tags. No Quadro 8 é apresentado um trecho de código do método `parseAsDicom`. Na linha 83 é verificado a direção da fatia para o tamanho e posicionamento da fatia ser realizado corretamente.

Cada objeto da classe `Shape` é adicionado ao objeto da classe `VolumeSlices`. Cada fatia é organizada baseando-se nos dados recuperados de cada uma. Após todas as fatias serem adicionadas pode ser feita a visualização volumétrica. Também o fatiamento do volume pode ser feito, mas somente na direção em que foram geradas as imagens do exame corrente.

Quadro 8 - Código do método `parseAsDicom`

```
42.   - (VolumeSlices *)parseAsDicom
43.   {
44.       DicomDecoder *dicomDecoder;
45.       int width, height;
46.       float depth;
47.       float x1, x2, y1, y2, z1, z2;
48.       GLint direction;
49.       NSString *orientation;
50.       FileInfo *fi;
51.       VolumeSlices *volume = [[VolumeSlices alloc] init];
52.
53.       for (NSString *file in self.dicomFiles) {
54.           dicomDecoder = [[DicomDecoder alloc] init:self.directory : file];
55.           if([dicomDecoder isDicom]) {
56.               [dicomDecoder decode];
57.               UIImage *img = [dicomDecoder getDicomImage];
58.
59.               depth = [[((NSMutableDictionary *)[dicomDecoder getValues])
60.                 objectForKey:[NSNumber numberWithInt:TAG_SLICE_THICKNESS]] floatValue];
61.
62.               [volume setThickness:(int)depth+1];
63.
64.               orientation = [((NSMutableDictionary *)[dicomDecoder getValues])
65.                    objectForKey:[NSNumber numberWithInt:TAG_ORIENTATION_SLICES]];
66.               NSArray *valueOrientation=[orientation componentsSeparatedByString:@"\\"];
67.               x1 = [[valueOrientation objectAtIndex:0] floatValue];
68.               y1 = [[valueOrientation objectAtIndex:1] floatValue];
69.               z1 = [[valueOrientation objectAtIndex:2] floatValue];
70.               x2 = [[valueOrientation objectAtIndex:3] floatValue];
71.               y2 = [[valueOrientation objectAtIndex:4] floatValue];
72.               z2 = [[valueOrientation objectAtIndex:5] floatValue];
73.
74.               fi = [dicomDecoder getFileInfo];
75.               width = [fi getWidth];
76.               height = [fi getHeight];
77.
78.               float thickness = depth+1;
79.               BOOL points = false;
80.
81.               Shape *slice = [[Shape alloc] initWithImage:img];
82.               [slice setOrientation: x1 :y1 :z1 :x2 :y2 :z2];
83.               direction = [slice calculateOrientation];
84.               [slice setThickness:depth];
85.               if (direction == SLICE_ORIENTATION_AXIAL) {
86.                   [slice setSizes:width :[volume getNextZPosition: thickness] :height];
87.               } else if (direction == SLICE_ORIENTATION_SAGITAL) {
88.                   [slice setSizes:[volume getNextZPosition: thickness] :height :width];
89.               } if (direction == SLICE_ORIENTATION_CORONAL) {
90.                   [slice setSizes:width :height :[volume getNextZPosition: thickness]];
91.               }
92.
93.               [slice generatePoints:points];
94.
95.               [volume addSlice:slice :file];
96.           }
97.       }
98.
99.       return volume;
100.  }
```

No método `generateOtherDirections`, na classe `VolumeSlices`, é gerada as imagens nas outras duas direções faltantes. Esse processo é feito com base nas imagens já existentes, que foram capturadas pelo equipamento médico. Nesse processo pode-se imaginar

um grande cubo dividido em partes iguais, em pequenos cubos, chamados cada um de `voxel`. Eles são gerados a partir de um `pixel` da imagem. A partir dos `voxels` gerados por todos os `pixels` do volume, cada um obtido de uma imagem em uma direção anatômica, pode-se obter imagens nas outras direções, que neste trabalho é duas. Como exemplo, podemos considerar que um aparelho de ressonância magnética tenha obtido as imagens de um exame na direção axial. Neste caso, o aplicativo irá gerar as imagens nas direções sagital e coronal com base nas imagens da direção axial.

Uma imagem bidimensional pode ser representada pela função de intensidade da luz `f(x, y)`, onde `x` e `y` representam as coordenadas espaciais e o valor de `f` em qualquer ponto `(x, y)` é proporcional ao brilho (ou níveis de cinza) da imagem naquele ponto.

Com base na descrição sobre geração das imagens nas outras direções, pode-se fazer a modelagem. A partir de um conjunto I_A de imagens na direção axial obtidas pelo aparelho de geração de imagens DICOM, precisa-se obter dois novos conjuntos de imagens I_S e I_C, sendo o primeiro o conjunto de imagens na direção sagital e o segundo na coronal. Desta forma o volume pode ser formado utilizando-se estes conjuntos, possibilitando o fatiamento nestas direções. No Quadro 9 pode ser visto como é obtido o conjunto de imagens na direção sagital a partir de um conjunto na direção axial.

Quadro 9 - Equação para gerar o conjunto na direção sagital de um conjunto na direção axial

$$Is = \sum_{\lambda=0}^{y} \sum_{\alpha=0}^{n} \sum_{\beta=0}^{x} S_\lambda(\alpha, \beta)$$

sendo $S_\lambda(\alpha, \beta) = f_\alpha(\beta, \lambda)$

onde:

y = número de colunas da imagem f_α

n = número de imagens no conjunto I_A

x = número de linhas da imagem f_α

Para obter-se o conjunto de imagens na direção coronal a partir do conjunto I_A, utiliza-se quase a mesma equação, com algumas alterações apenas. Cada imagem c_β gerada pertencente ao conjunto I_C, deve ser obtida com $c_\beta(\alpha, \beta)$ igual a $f_\alpha(\beta, \gamma)$. O trecho do código que gera as imagens na direção coronal a partir do conjunto de imagens na direção axial pode ser visto no Quadro 10.

Quadro 10 - Trecho do código do algoritmo que gera as imagens na direção coronal

```
for (int k = 0; k < axialCount; k+=thickness) {
   Shape *slice = [self.slicesAxial objectAtIndex:k];
   UIImage *img = slice.imageTexture;

   CGImageRef inImage = img.CGImage;

   CFDataRef dataRefImg;
   int rows = img.size.height;
   int cols = img.size.width;
   int numGenerateSlice = rows/NUM_GENERATE_SLICES;
   t = numGenerateSlice;
   int space = (rows-temp)/VAR_ADJUST;
   if(temp >= rows) {
       space = 0;
       temp = rows;
   }
    dataRefImg = CGDataProviderCopyData(CGImageGetDataProvider(inImage));
    UInt32 * pixelBufImg = (UInt32 *) CFDataGetBytePtr(dataRefImg);

    for (int n = 0, x = 0; n < rows; n+=numGenerateSlice, x++) {
            Shape *newSlice;
            if(!create && (k == 0)) {
                newSlice = [slice copy];

                UInt32 *pixelBufNewImg = malloc((space + temp) * cols * (sizeof(UInt32)));
                newSlice.imageTexture = [self buff32BitsToUIImage:inImage
      :pixelBufNewImg :cols :(space + temp)];

                [newSlice setOrientation:SLICE_ORIENTATION_CORONAL];
                [newSlice setThickness:t];
                [newSlice setSizes:cols :(temp+space) :nextZCoronalPosition];
                [newSlice generatePoints: points];
                nextZCoronalPosition += t;
                [self.slicesCoronal addObject:newSlice];
            } else {
                newSlice = [self.slicesCoronal objectAtIndex:x];
            }

            UIImage *newImg = newSlice.imageTexture;

            CGImageRef inNewImage = newImg.CGImage;

            CFMutableDataRef dataRefNewImg;
            int colsNewImg = newImg.size.width;
            dataRefNewImg = CFDataCreateMutableCopy(0, 0,
      CGDataProviderCopyData(CGImageGetDataProvider(inNewImage)));

            UInt32 *pixelBufNewImg = (UInt32 *) CFDataGetMutableBytePtr(dataRefNewImg);

            for (int i = 0; i < cols; i++) {
                  uint8_t *rgbaPixel = (uint8_t *) &pixelBufImg[cols*n + i];
                  uint8_t *rgbaPixelNew = (uint8_t *)
      &pixelBufNewImg[colsNewImg*(k*thickness+space) + i];
                  int t0 = rgbaPixel[0];
                  int t1 = rgbaPixel[1];
                  int t2 = rgbaPixel[2];
                  int t3 = rgbaPixel[3];

                  rgbaPixelNew[0] = t0;
                  rgbaPixelNew[1] = t1;
                  rgbaPixelNew[2] = t2;
                  rgbaPixelNew[3] = t3;
              }

            newSlice.imageTexture = [self buff32BitsToUIImage:inNewImage :pixelBufNewImg
      :CGImageGetWidth(inNewImage) :CGImageGetHeight(inNewImage)];
    }
}
```

3.3.2.3 Visualização em duas dimensões

Nesta etapa não é necessário mais processamento dos dados das imagens DICOM. As

fatias são passadas para a classe `Viewer2DController`. Nesta classe está o controle de qual imagem será visualizada. As imagens são organizadas em uma pilha. Cada imagem é visualizada uma de cada vez. O usuário pode utilizar um *slider* para poder alterar a imagem corrente visualizada. No Quadro 11 está o código do método `valueSliderChange`, onde é atribuída a imagem corrente a ser visualizada.

Quadro 11 - Código do método que verifica qual imagem será visualizada

```
74. - (IBAction)valueSliderChange:(id)sender {
75.       NSMutableArray *current;
76.
77.       indexSlider = [(UISlider*)sender value]*viewer2D.slices.count;
78.       indexSlider--;
79.       current = viewer2D.slices;
80.
81.       if (indexSlider < 0) {
82.          indexSlider = 0;
83.       }
84.       _currentImage = [current objectAtIndex:indexSlider];
85.       [self updateImage];
86. }
```

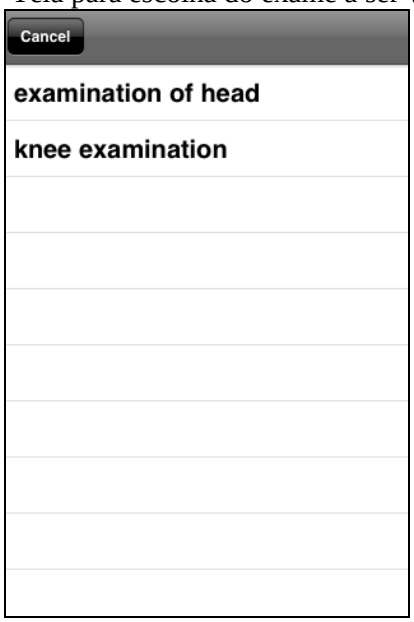
3.3.3 Operacionalidade da implementação

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

3.3.3.1 Visualização em três dimensões

Ao usuário executar a aplicação será aberta a tela de visualização volumétrica. Logo após a tela ser aberta é feito o redirecionamento para outra tela, onde será escolhido o exame a ser visualizado pelo usuário. Esta tela pode ser visualizada na Figura 18.

Figura 18 - Tela para escolha do exame a ser visualizado



O usuário escolhe um dos exames. Em cada um deles existem vários arquivos `dcm` que são lidos para posteriormente poder realizar o processamento dos dados. Ao realizar a escolha, a execução do aplicativo retorna para a tela de visualização volumétrica. Neste momento é desenhado na tela o volume do exame escolhido. Na Figura 19 é apresentada a tela.

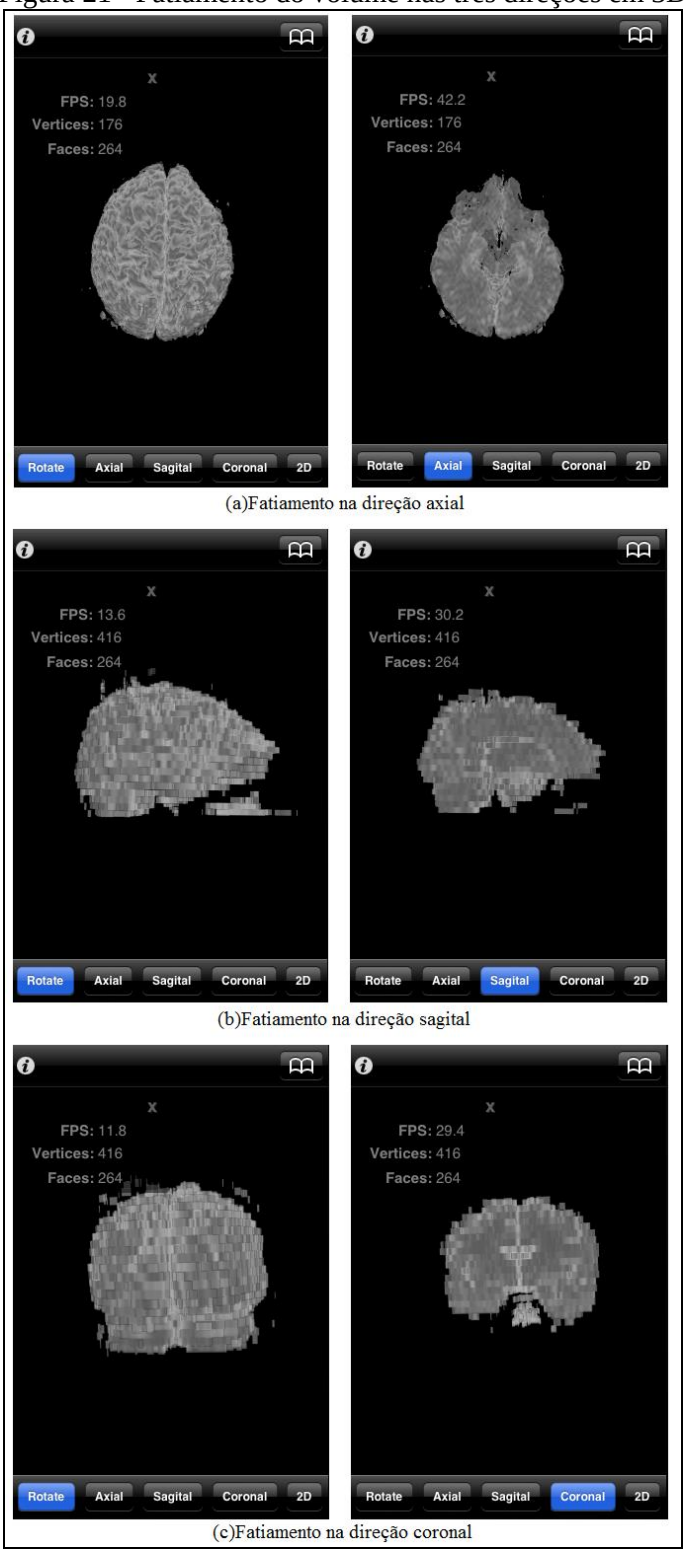
Figura 19 - Tela para visualização em três dimensões

Ao dar um toque no centro da tela, as barras com botões na parte superior e inferior da tela são ocultadas. Na parte inferior da tela de visualização volumétrica pode-se observar que existem cinco botões. Ao desenhar o volume o botão `Rotate` fica selecionando. Quando ele está selecionado o usuário pode rotacionar livremente o volume. Na Figura 20 pode ser observado um exame em três posições diferentes após sofrer rotações.

Figura 20 - Rotação do volume

Há três botões ao lado do `Rotate`, que são: `Axial`, `Sagital` e `Coronal`. Ao selecionar o botão `Axial` o usuário poderá fatiar o volume no sentido axial (Figura 21a). Quando `Sagital` é selecionado, pode-se fatiar o volume na direção sagital (Figura 21b). Por fim, quando seleciona-se o botão `Coronal`, o usuário pode fatiar o volume na direção coronal (Figura 21c).

Figura 21 - Fatiamento do volume nas três direções em 3D



3.3.3.2 Visualização em duas dimensões

Na tela de visualização em três dimensões, existe um botão chamado `2D`. Quando o usuário selecionar este botão, a aplicação é direcionada para a tela de visualização das imagens em duas dimensões. Nesta tela de visualização 2D há um `slider`, onde o usuário pode alterar a imagem que está sendo mostrada. A Figura 22 ilustra esta tela.

Figura 22 - Tela de visualização em duas dimensões

3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta uma aplicação para a visualização volumétrica de imagens DICOM para iOS. Nesta etapa primeiramente são mostrados os resultados do volume gerado a partir das imagens recuperadas dos arquivos lidos. Posteriormente são comparados os resultados deste trabalho com os trabalhos correlatos. Por fim é discutido o consumo de memória da aplicação e desempenho.

3.4.1 Testes da geração do volume

Na etapa de leitura dos arquivos DICOM, primeiramente tentou-se utilizar bibliotecas já existentes. Não foi encontrada nenhuma biblioteca para leitura de arquivos `dcm` pronta para

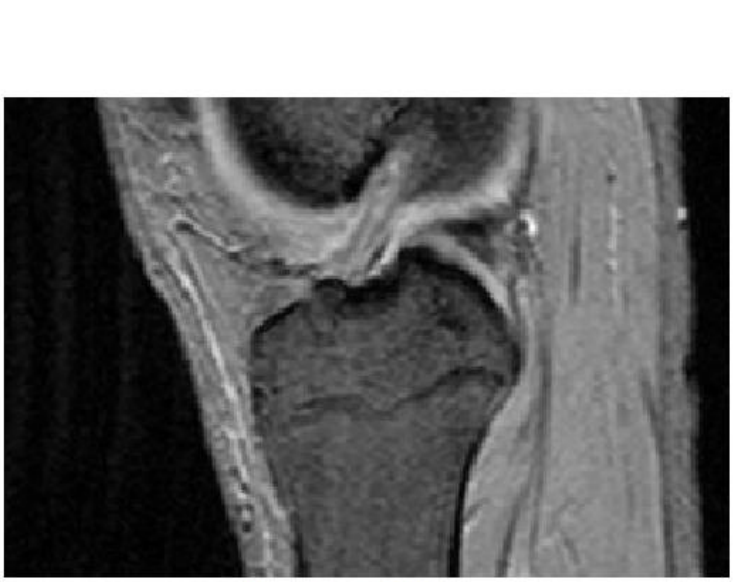
ser utilizada no desenvolvimento de aplicações para iOS. No primeiro momento buscou-se utilizar, sem sucesso, bibliotecas desenvolvidas em C/C++. Não obteve-se sucesso nas tentativas de compilar tais bibliotecas no projeto de desenvolvimento da aplicação, visto que elas não foram desenvolvidas com o objetivo de serem executas no iOS. Posteriormente tentou-se utilizar bibliotecas usadas no desenvolvimento de aplicativos para Mac OS. Novamente sem sucesso no desenvolvimento do projeto da aplicação. Desta vez o principal problema foi trabalhar com as dependências das bibliotecas, que possuem restrições que não permitiram serem utilizadas no projeto.

Desta forma optou-se por desenvolver um leitor de imagens DICOM. Este leitor lê imagens `dcm` e recupera a imagem de cada arquivo, além de recuperar informações importantes para a organização de cada fatia na formação do volume. O leitor é limitado a ler aquivos com imagens que possuem os `pixels` representados em dezesseis bits com sinal ou sem sinal. O seu desenvolvimento foi baseado no ImageJ (2013). Um tempo maior do que o planejado foi gasto nesta etapa, pois esperava-se utilizar alguma ferramenta pronta para esta etapa mas não foi possível. O padrão DICOM é extenso e não é algo simples desenvolver uma ferramenta para ler todos os tipos de arquivos possíveis. Como a proposta deste trabalho é a visualização volumétrica dos dados, não foi aplicado mais esforço no desenvolvimento do leitor, sendo suficiente conseguir ler um número restrito de exames para ter os dados para a etapa de visualização.

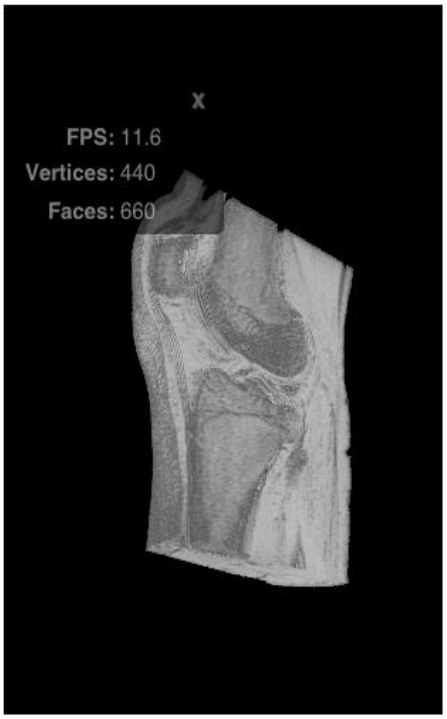
Para a geração do volume, utilizou-se as imagens DICOM recuperadas pelo leitor. O volume foi criado com várias fatias. Cada fatia recebeu como textura uma imagem lida. Dados obtidos em cada arquivo foram utilizados na organização das fatias no volume. Para que os `pixels` que não fazem parte do corpo não fossem desenhados, utilizou-se um limiar para determinar que `pixel` deve estar no volume. Com o OpenGL ES, pode-se alterar as configurações da opacidade no momento em que são desenhados os objetos. Desta forma, o limiar divide os `pixels` que fazem parte do corpo dos que não fazem parte. Os que fazem parte do corpo recebem o valor máximo para a opacidade, enquanto os outros recebem valor mínimo. Sendo assim, ao desenhar o volume, o OpenGL ES vai desenhar somente os `pixels` que fazem parte do corpo. Na Figura 23a, observa-se toda a imagem de um exame de ressonância magnética do joelho em uma visualização 2D. Pode-se observar alguns ruídos na parte preta da figura, mas estes `pixels` não possuem valor suficiente para aparecerem no volume e na Figura 23b tem-se a visualização volumétrica onde os `pixels` escuros não fazem parte do volume.

No arquivo `dcm` existe uma `tag` chamada *spacing between slices*, onde se encontra o valor da distância entre uma fatia e outra no momento da obtenção dos dados pelo aparelho de resonância magnética. Para obter-se o tamanho real da imagem, organizou-se cada fatia a uma distância uma da outra. O valor desta distância é obtido na `tag` *spacing between slices*. Desta forma, ao visualizar o volume em certas posições, observa-se um espaço entre as fatias que não foi tratado por este trabalho. A Figura 24 ilustra este comportamento. Quanto maior a quantidade de fatias do volume menor é o espaço entre elas, assim este efeito terá sua percepção reduzida. Ainda na Figura 24 observa-se a diferença entre a visualização do exame do crânio e do joelho. No exame do joelho (direita) há 55 fatias enquanto que no exame do crânio (esquerda) há 22 fatias.

Figura 23 - Ruído na imagem 2D e visualização volumétrica sem ruídos



(a)Imagem com ruídos



(b)Visualização volumétrica sem ruídos

Na geração das imagens em outras direções observa-se dois comportamentos também relacionados com a quantidade de fatias que possui o exame. Quanto maior a quantidade de fatias melhor vai ser a resolução da imagem gerada. Na Figura 25 é apresentada uma comparação entre imagens geradas por 22 fatias e 55 fatias.

O próximo passo é a geração dos `voxels` para a aplicação de um algoritmo de visualização volumétrica, que seria o *ray casting*. Não foi possível implementar este algoritmo neste trabalho devido ao tempo, no início do trabalho houve dificuldade relacionada a plataforma e a linguagem Objective-C, que é obrigada no desenvolvimento para iOS.

Estudos e testes foram feitos sobre este algoritmo utilizando a linguagem de programação em Java. Mais informações podem ser encontradas no APÊNDICE A.

Figura 24 - Espaços entre as fatias

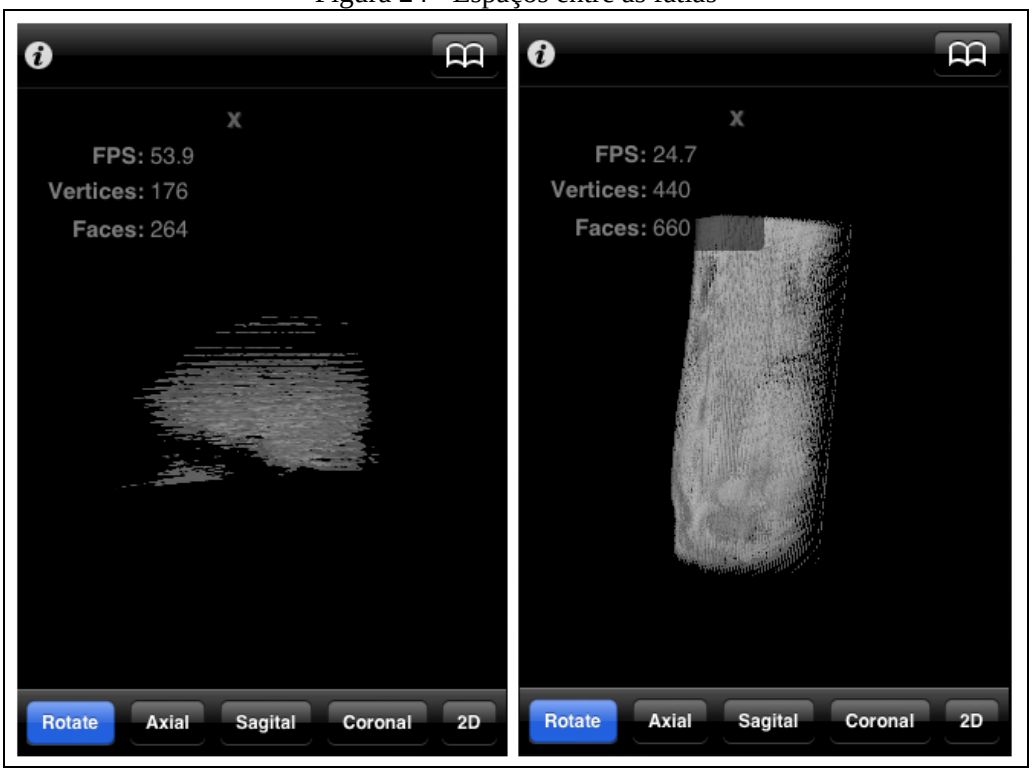
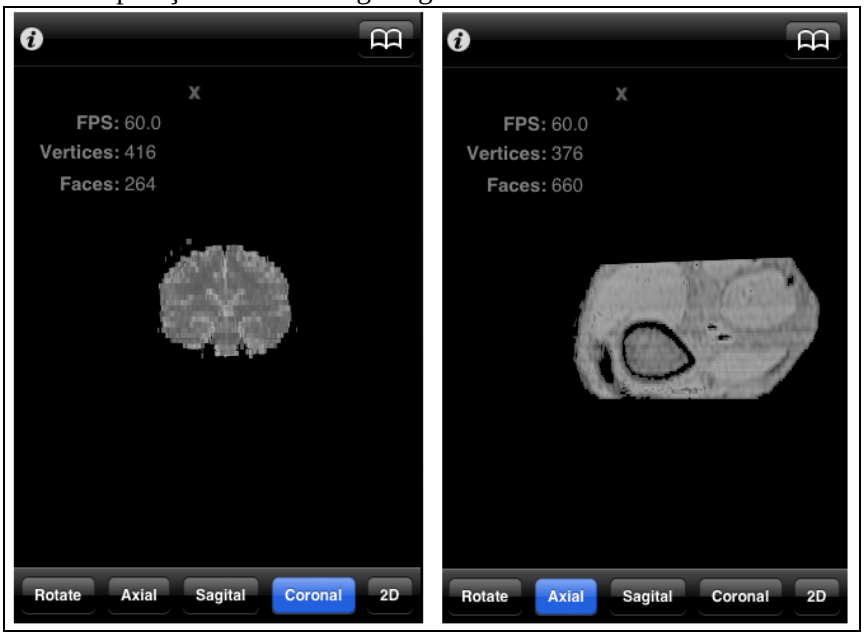


Figura 25 - Comparação entre as imagens geradas com o exame de 22 fatias e 55 fatias



3.4.2 Comparação com os trabalhos correlatos

Nesta seção são comparadas as principais características deste trabalho com as dos trabalhos correlatos. No Quadro 12 é apresentado um comparativo entre este trabalho e os trabalhos correlatos.

Ao observar o quadro, conclui-se que o Osirix é a aplicação mais completa, possuindo todas as características principais observadas. Segundo Osirix (2012), a aplicação foi resultado de mais de cinco anos de pesquisas e desenvolvimento em imagens digitais. Além da aplicação para *desktop*, existe também o Osirix para dispositivos móveis, na plataforma iOS. Pode haver a comunicação entre a aplicação móvel e a aplicação *desktop*. No Osirix, mesmo a visualização volumétrica sendo utilizando o *ray casting*, o fatiamento pode ser feito no volume. Além disso o fatiamento pode ser feito em qualquer direção.

Quadro 12 – Comparação entre este trabalho com os trabalhos correlatos

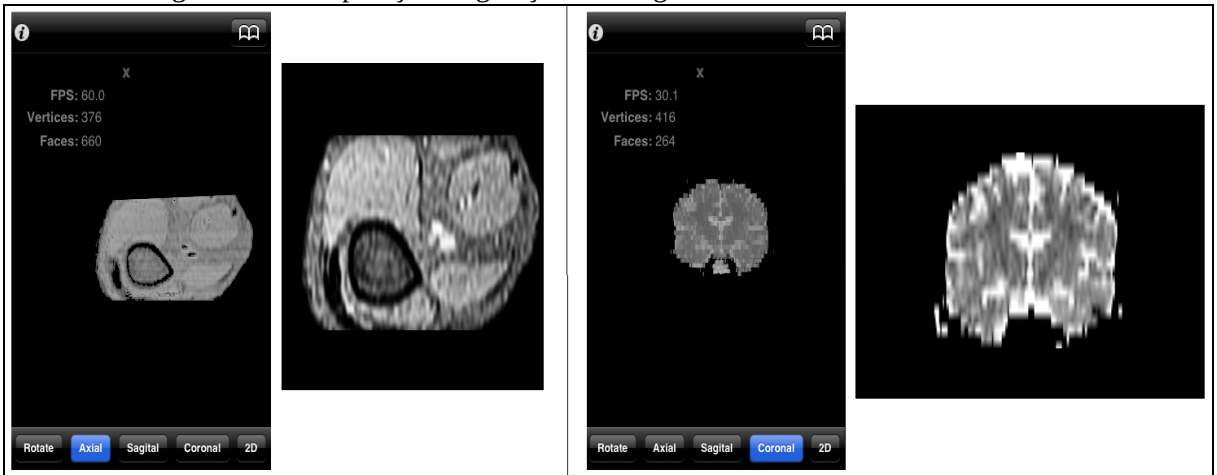
Características\Trabalhos correlatos	InVesalius	Osirix	Roepke (2010)	Oliveira (2013)
Ler arquivos DICOM	X	X		X
Visualização em duas dimensões	X	X	X	X
Tratamento das imagens	X	X	X	
Visualização volumétrica	X	X		X
Visualização interna do volume	X	X		
Visualização em mais de três dimensões		X		
Separação do volume em componentes	X	X		
Fatiamento do volume	X	X		X
Geração da imagem em mais de uma direção anatômica	X	X		X
Aplicação para dispositivos móveis		X	X	X

O InVesalius vem evoluindo em mais de uma década, sendo atualmente uma aplicação bem completa. Porém não possui uma versão para dispositivos móveis, como pode ser observado no Quadro 12. No InVesalius também é possível fazer o fatiamento somente em três dimensões, como neste trabalho. Pode-se visualizar nas Figura 26a e Figura 26b a comparação de imagens geradas no InVesalius e neste trabalho. Observa-se que o resultado obtido neste trabalho é semelhante ao do InVesalius. A visualização volumétrica foi feita utilizando o *ray casting*. Desta forma, pode-se separar em componentes o volume, como visualizar somente o osso, ou a gordura, ou os vasos sanguíneos. Por outro lado, não é possível fazer o fatiamento no volume. Na aplicação desenvolvida neste trabalho, é possível fazer o fatiamento do volume, porém o algoritmo *ray casting* não foi implmentado, não sendo possível separar a visualização do volume em componentes.

O aplicativo desenvolvido por Roepke (2010) é resultado de um trabalho acadêmico, no qual a principal característica para com este trabalho é o tratamento das imagens, podendo-se alterar brilho e contraste. Porém, em seu trabalho Roepke (2010) não lê as imagens de um arquivo `dcm`. Ele lê arquivos JPEG utilizando uma ferramenta disponibilizada pela Apple. Na aplicação desenvolvida neste trabalho as imagens são lidas do arquivo `dcm`. Além disso é feita

a visualização volumétrica das imagens e o fatiamento do volume gerado em três direções anatômicas, além da visualização 2D das imagens, também disponível no trabalho de Roepke (2010).

Figura 26 - Comparação da geração de imagens no InVesalius e neste trabalho



(a)Joelho - comparação entre este trabalho e o InVesalius

(b)Crânio - comparação entre este trabalho e o InVesalius

3.4.3 Memória e desempenho

Para fazer os testes de desempenho e de memória da aplicação foram utilizados dois exames médicos obtidos por ressonância magnética, um do crânio com 22 fatias e outro do joelho com 55 fatias. As imagens lidas dos aquivos `dcm` do exame do crânio possuem tamanho de 256x256, enquanto que as lidas do exame do joelho possuem tamanho de 512x512. A aplicação foi executada em um dispositivo iPad 4 com 1 GB de memória DDR2 RAM, processador dual core Apple Swift de 1.4 GHz e processador gráfico Quad-core PowerVR SGX554MP4. Foi utilizado o programa Instruments do pacote de ferramentas do Xcode para obter os dados que são analizados.

3.4.3.1 Consumo de memória

Primeiramente será analisado o uso da memória na visualização dos dois exames. A Figura 27 ilustra a visualização do exame quando o mesmo é carregado.

Ao selecionar na `lista de exames` um exame a aplicação inicia a leitura dos arquivos. Neste momento o uso de memória chega aos seus máximos de utilização. Após terminar a leitura o uso de memória é estabilizado. O Quadro 13 apresenta uma comparação do uso de memória em cada momento na visualização do exame do crânio, que possui 22 fatias. Nestas análises chamou-se de momento um intervalo de tempo que a aplicação executa alguma ação.

Figura 27 - Primeiro momento da visualização do joelho

Quadro 13 - Uso da memória por evento na visualização do exame do crânio

Momento	Uso inicial de memória	Uso mínimo de memória	Uso máximo de memória
Antes da seleção do exame	62,98MB	32,91MB	62,98MB
Ao ler os arquivos do exame	33,03MB	33,03MB	86,79MB
Visualização volumétrica do exame	43,46MB	43,46MB	43,46MB

Observa-se que o uso máximo de memória ocorre quando estão sendo lidos os arquivos do exame. Após este momento o uso de memória se estabiliza em um valor menor, utilizamdo 32,91MB. Inicialmente, no desenvolvimento da aplicação, as imagens utilizadas no fatiamento nas duas direções que não possuem no exame, eram geradas logo após o momento em que os arquivos eram lidos. O algoritmo para gerar estas imagens executou somente no simulador, pois no dispositivo o uso de memória excedeu o permitido. As imagens são geradas em uma determinada direção somente quando o usuário solicita o fatiamento nesta direção. No Quadro 14 é apresentada a comparação do uso de memória em cada momento na visualização do exame do joelho, que possui 55 fatias.

Quadro 14 - Uso da memória por evento na visualização do exame do joelho

Momento	Uso inicial de memória	Uso mínimo de memória	Uso máximo de memória
Antes da seleção do exame	62,98MB	32,91MB	62,98MB
Ao ler os arquivos do exame	49,87MB	49,87MB	324,11MB
Visualização volumétrica do exame	131,07MB	131,07MB	131,03MB

Observa-se ao comparar o quadro dos dois exames que na leitura dos arquivos do exame do joelho o uso de memória é muito superior. O valor chega a ser de aproximadamente quatro vezes mais. Isto pode ser explicado pela quantidade maior de imagens e pelo tamanho maior das imagens, pois no exame do joelho há aproximadamento o dobro de fatias e a imagem recuperada de cada uma das fatias é duas vezes maior do que as do exame do crânio. Após a leitura dos arquivos e a geração do volume, o volume é renderizado. Neste momento o uso de memória se estabiliza. Neste momento somente os valores referentes ao dempenho se alteram, conforme o usuário vai manipulando o volume.

Foram feitos testes utilizando o simulador para verificar o desempenho na geração das imagens utilizadas no fatiamento do volume que não existem no exame, o uso da memória volta a subir quase chegando ao limite. A utilização do simulador foi necessária pois o uso de memória utrapassou o limite na execução no dispositivo. No Quadro 15 é apresentada uma comparação entre a geração das imagens no exame do crânio e do joelho. Como a quantidade de imagens no exame do joelho é maior, além de ter imagens com o dobro do tamanho, seu gasto de memória é o maior alcançado nos testes realizados.

Quadro 15 - Uso da memória na geração das imagens no simulador

Exame	Uso inicial de memória	Uso máximo de memória
Crânio	64,06MB	87,48MB
Joelho	203,03MB	315,67MB

Observa-se olhando o uso inicial de memória, que o uso de memória tem uma variação considerável entre a análise feita utilizando o dispositivo e a análise feita utilizando o simulador. Esta análise pode ser feita comparando o uso de memória na visualização volumétrica no Quadro 14 para o joelho, onde é apresentado 131.07MB na visualização volumétrica e o uso inicial de memória na geração das imagens, onde os dados foram obtidos no simulador, é de 203.03MB. E no Quadro 13 para o crânio, onde é mostrado 43.46MB na visualização volumétrica e o uso inicial de memória na geração das imagens, do qual os dados foram obtidos no simulador, é de 64.06MB. O gasto de memória na visualização volumétrica é o mesmo do uso inicial de memória na geração de imagens, esta diferença observada entre os quadros se deve por obter os dados da geração utilizando o simulador.

3.4.3.2 Desempenho

A avaliação do desempenho foi feita da mesma forma, analisando e comparando o tempo gasto nos dois exames. O algoritmo de geração de imagens foi testado no simulador, pois ao executar o algoritmo de geração de arquivos o uso de memória no dispositivo excede

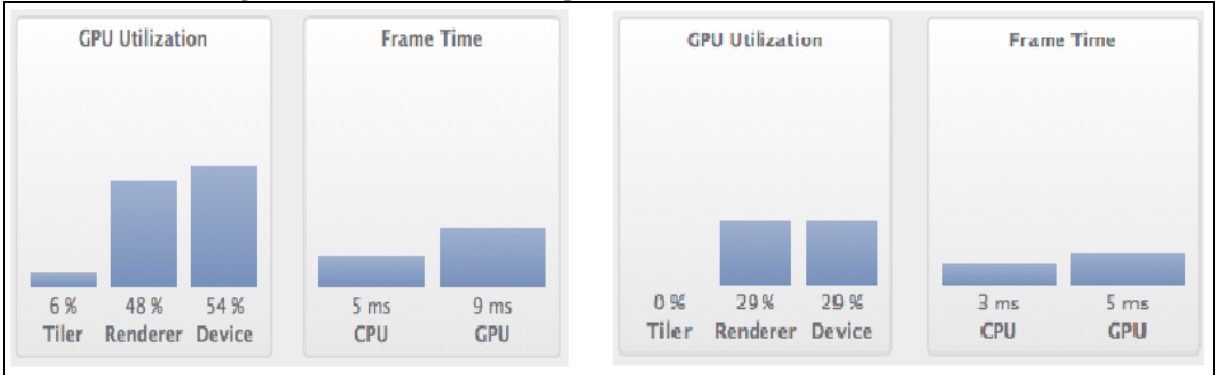
o limite possível. Há uma diferença de até sete vezes entre dados obtidos no dispositivo com os dados obtidos no simulador. No Quadro 16 é ilustrado o desempenho por ação na visualização dos dois volumes.

Quadro 16 - Desempenho por momento na visualização dos dois exames

Ação	Tempo (segundos) do exame do crânio	Tempo (segundos) do exame do joelho
Ler os arquivos do exame e desenhar o volume	5,618	14,207
Gerar as imagens (simulador)	0,758	3,07

Na Figura 28 pode ser observado 4 gráficos obtidos com o Xcode. Estes gráficos mostram o consumo de CPU e GPU na visualização volumétrica do exame do joelho e no exame da cabeça. Os dados mostram que o consumo chega a ser mais de duas vezes maior na visualização do exame do joelho.

Figura 28 – Gráfico do desempenho obtido com recursos do Xcode



(a)Visualização volumétrica do joelho (b)Visualização volumétrica da cabeça

4 CONCLUSÕES

Neste trabalho apresentou-se o desenvolvimento de um aplicação de visualização volumétrica de imagens DICOM para a plataforma iOS. Os resultados dos testes realizados mostram que o aplicativo atende aos objetivos propostos.

Este trabalho possui uma contribuição importante com o desenvolvimento, mesmo que parcial, de uma biblioteca para leitura de arquivos DICOM para iOS. O que torna este ponto mais relevante é o fato de não ter sido possível adaptar nenhuma biblioteca disponível em C/C++ e em Objective-C para iOS, sendo que esta adaptação poderia levar um tempo muito elevado e posteriormente não ter sucesso.

O volume gerado para a visualização foi obtido pela organização das fatias no espaço 3D, utilizando-se as imagens como textura em cada fatia. Por um lado, este fato permite o fatiamento do volume, ou seja, na visualização volumétrica permite-se fazer o fatiamento do volume em uma direção a cada vez. Mas, como não foi implementado o *ray casting*, não é possível visualizar elementos internos e nem separar a visualização por elementos. Por exemplo, não é possível visualizar somente os ossos do paciente.

Como o volume foi formado por um conjunto de fatias e cada fatia encontra-se a uma determinada distância das outras, então em determinadas posições da visualização é possível observar estes espaços. Na geração das imagens nas duas direções não existentes no exame corrente, o resultado foi semelhante ao InVesalius, onde a qualidade das imagens geradas também depende da quantidade de imagens no exame original.

Um desafio no início do desenvolvimento do trabalho foi a linguagem Objective-C. Esta linguagem é obrigatória no desenvolvimento de aplicativos para esta plataforma. Foi bom para o aprofundamento nesta linguagem e plataforma.

4.1 EXTENSÕES

Após o desenvolvimento deste trabalho pode-se detectar pontos de melhoria e continuidade do mesmo. Eles são:

a) desenvolver a biblioteca para ler arquivos DICOM, visto que neste trabalho somente imagens de 16 bits com sinal são lidas. Desta forma qualquer arquivo DICOM poderia ser lido neste trabalho. Outras aplicações que não trabalhem com as imagens, mas sim com as informações do paciente que também podem ser obtidas lendo o arquivo `dcm`, poderiam utilizar também esta biblioteca;
b) otimizar o uso de memória para que todos os recursos da aplicação possa ser utilizada no dispositivo físico;

c) desenvolver um aplicativo para trabalhar desenhar o contorno do volume. Os pontos externos do volume poderiam ser obtidos e consequentemente a superfície do corpo poderia ser desenhada na tela de visualização volumétrica;
d) implementar o algoritmo de *ray casting* para realizar a visualização volumétrica, desta forma seria possível fazer outros processamentos na imagem, como detectar quais `pixels` representam o osso do paciente;
e) implementar o algoritmo de *marching cubes* para realizar a visualização volumétrica. Também teria a possibilidade de trabalhar com os elementos do corpo que podem ser observados na imagem;
f) realizar o fatiamento em outras direções anatômicas. Neste trabalho o fatiamento é feito em três direções, axial, coronal e sagital. Na aplicação para Mac OS do Osirix, por exemplo, o usuário pode ao visualizar o volume informar onde será feito o corte para fazer o fatiamento do volume;
g) realizar testes para explorar mais informações sobre o limite físico de executar estes algoritmos em dispositivos móveis.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLE INC. **iOS developer library.** [S.l.], 2012. Disponível em: <http://developer.apple.com/library/ios/navigation/>. Acesso em: 19 set. 2012.

DOMETERCO, José H. **Uma plataforma para a visualização tridimensional de regiões de ativação cerebral por imagens de ressonância magnética funcional.** 2002. 89f. Dissertação (Mestrado em Informática) - Curso de Pós-graduação em Informática, Universidade Federal do Paraná, Curitiba.

FRANCESCHI, Wagnes B. **Procedimentos e práticas para digitalização de imagens médicas.** 2006. 144 f. Dissertação (Mestrado em Ciências) - Curso de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná, Curitiba.

FREITAS, Cilas. **Uma arquitetura baseada em padrões abertos para a visualização científica via internet aplicada à medicina.** 2002. 84 f. Dissertação (Mestrado em Informática) - Curso de Pós-graduação em Informática, Universidade Federal do Paraná, Curitiba.

IMIANOWSKY, Felipe A. **iOBJ**. Blumenau, 2013. Disponível em: <https://github.com/felipowsky/iOBJ>. Acesso em: 4 jun. 2013.

IMAGEJ. [S.l.], 2013. Disponível em: <http://rsbweb.nih.gov/ij/>. Acesso em: 4 jun. 2013.

INVESALIUS.- [S.l.], 2012. Disponível em: <http://svn.softwarepublico.gov.br/trac/invesalius/wiki/WikiStart>. Acesso em: 19 set. 2012.

KHRONOS GROUP. **OpenGL ES overview.** [S.l.], 2012. Disponível em: <http://www.khronos.org/opengles/>. Acesso em: 19 set. 2012.

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. **DICOM viewer.** Santa Monica, 2012. Disponível em: <http://www.osirix-viewer.com/>. Acesso em: 19 set. 2012.

PAIVA, Anselmo C.; SEIXAS, Roberto B.; GATTASS, Marcelo. **Introdução à visualização volumétrica.** 1999. 107 f. Monografia em Ciência da Computação (Bacharelado em Ciência da Computação), Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

PRAUCHNER, João L. **Especificação de funções de transferência para visualização volumétrica.** 2005. 64 f. Dissertação (Mestrado em Ciência da Computação) - Curso de Pós-graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

ROEPKE, Marwin. **Visualizador de imagens radiológicas 2D para Iphone.** 2010. 51 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

RUBIO, Cássio A. **Estilização e visualização tridimensional de tumores intracranianos em exames de tomografia computadorizada.** 2003. 108 f. Dissertação (Mestrado em Informática) - Curso de Pós-graduação em Informática, Universidade Federal do Paraná, Curitiba.

SEIXAS, Roberto B. **Visualização volumétrica com ray casting em um ambiente distribuído.** 1997. 64 f. Tese (Doutorado em Ciências em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

SILVA, Isabel C. **Avaliação da qualidade de imagens médicas geradas por ray casting.** 2003. 125 f. Dissertação (Mestrado em Ciência da Computação) - Curso de Pós-graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

APÊNDICE A – Algoritmo de visualização volumétrica ray casting

Neste apêndice são apresentadas mais detalhes sobre o algoritmo ray casting. Este algoritmo é baseado em leis da física para a propagação da luz em materiais coloridos e semitransparentes. Ele consiste no lançamento de um raio a partir de cada `pixel` do plano de visão. Cada raio disparado poderá atingir os volumes que estejam no espaço volumétrico. Quando um raio atinge um volume, as opacidades e cores são acumuladas conforme o raio passa através do volume. O raio pode ser representado pela equação apresentada no Quadro 17.

Quadro 17 – Equação de um raio

$$R(t) = R_0 + R_d t$$

Sendo:
R_0 - ponto de origem do raio
R_d - direção do raio
t - parâmetro da equação paramétrica da reta

Implementou-se um algoritmo para lançar um raio e verificar sua intersecção com um volume. Para verificar se o raio penetra no volume, na implementação feita em java foi utilizada a equação paramétrica da intersecção de uma reta no plano, que é apresentada no Quadro 18. A implementação do método que obtém o parâmetro `t` pode ser vista no Quadro 19.

Quadro 18 – Equação paramétrica da intersecção de um raio com um plano

$$t_i = -\frac{N \bullet R_0 + D}{N \bullet R_d}$$

Sendo:
N - normal do plano verificado
D - coeficiente do plano
R_0 - ponto de origem do raio
R_d - direção do raio
$\bullet$ - produto escalar

Quadro 19 – Método que obtém o valor de `t`

```
36.     public double getTParameter(Ray ray) {
37.        point.subtract(ray.getOriginVector());
38.        double t = point.dot(normal) / (ray.getDirection().dot(normal));
39.
40.        return t;
41.     }
```

Para obter o ponto de intersecção deve-se substituir o valor de `t` na equação do Quadro 17 pelo valor obtido na equação do Quadro 18.