UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISUALIZADOR DE IMAGENS RADIOLÓGICAS 2D PARA IPHONE

MARWIN ROEPKE

BLUMENAU
2010

2010/1-17

MARWIN ROEPKE

VISUALIZADOR DE IMAGENS RADIOLÓGICAS 2D PARA IPHONE

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

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

BLUMENAU
2010

2010/2-17

VISUALIZADOR DE IMAGENS RADIOLÓGICAS 2D PARA IPHONE

Por

MARWIN ROEPKE

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

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

Blumenau, 01 junho 2010

Dedico este trabalho a todos os amigos, familiares e colegas, por entenderem que nem sempre estava disponível para ajudá-los neste período.

AGRADECIMENTOS

À meus pais pelo apoio.

Aos meus amigos, pela paciência, em especial aos que sempre estiveram ao meu lado para ouvir mesmo sem entender o assunto.

Ao meu orientador, Dalton Solano dos Reis, por ter me ajudado em todas as necessidades e pela atenção. E a todos que direta ou indiretamente me apoiaram e ajudaram para que este trabalho fosse possível.

Enquanto suspiramos por uma vida sem dificuldades, devemos nos lembrar que o carvalho cresce forte através de ventos contrários e que os diamantes são formados sob pressão.

Peter Marshall

RESUMO

Este trabalho apresenta a implementação de um visualizador de imagens médicas para o iPhone OS, desenvolvida usando frameworks UIKit, Foundation, Quartz Core e Core Graphics disponibilizadas pela Apple. Este visualizador permite que se acesse imagens JPEG obtidas através de arquivos DICOM de uma URL. Com as imagens carregadas é possível adicionar os efeitos de brilho e contraste, bem como marcar regiões da imagem para analises posteriores enviando-as por e-mail.

Palavras-chave: iPhone. DICOM. Imagens médicas.

ABSTRACT

This paper describes the implementation of an Medical image viewer for the iPhone OS, developed using the frameworks UIKit, Foundation, Core Graphics and Quartz Core available from Apple. This viewer lets you access JPEG images obtained through DICOM files from an URL. With the uploaded images can be added effects of brightness, contrast, and mark regions of the image to later analysis sending it by e-mail.

Key-words: iPhone. DICOM. Medical imaging.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

3D – Três Dimensões

DICOM - *Digital Imaging and COmmunications in Medicine*

JPEG – *Joint Photografic Experts Group*

OpenGL - *Open Graphics Library*

SUMÁRIO

1 INTRODUÇÃO

Segundo Sabbatini (1999, p. 22-23), “A partir da invenção do tomógrafo computadorizado, até o desenvolvimento da telemedicina pelos cientistas da NASA, aconteceu um aumento nos produtos biomédicos, é onde entra a área de informática”.

Com o avanço da tecnologia e da informática, os profissionais da área médica já podem usufruir dos notebooks como ferramenta para a visualização de imagens radiológicas. Outra ferramenta são os novos dispositivos móveis, como o iPhone, pois possuem tecnologia suficiente para atender a necessidade básica de visualização de imagens 3D, trazendo assim a possibilidade dos profissionais da área médica terem o acesso a imagens. A possibilidade dos profissionais contarem com a mobilidade de um iPhone faz-se importante quando comparada com uma estação de trabalho.

No momento atual, na sua maioria, para terem acesso a imagens radiológicas os profissionais da área médica necessitam de uma estação de trabalho com um software específico de visualização de imagens no formato *Digital Imaging and COmmunications in Medicine* (DICOM).

Contando com um visualizador de imagens 3D em um dispositivo móvel, o médico pode efetuar uma análise das imagens independentemente do horário ou local em que ele se encontre, podendo assim melhorar o atendimento aos pacientes.

Segundo Melo (2009), “os doentes estão a cada dia mais impacientes quanto ao tratamento recebido nos consultórios, hospitais e unidades de saúde”. Efetuar a análise de imagens radiológicas e iniciar um tratamento ao pé da cama de um paciente faz com que ele tenha mais confiança no médico. Para ajudar neste aspecto de contato entre os pacientes e os médicos, o visualizador de imagens radiológicas num dispositivo móvel auxiliaria.

Com o exposto acima, este trabalho pretende estudar o padrão de imagens DICOM e desenvolver uma ferramenta que possibilite a visualização de imagens JPEG obtidas das imagens DICOM, possibilitando ao usuário interagir com estas imagens. Serão utilizados os recursos oferecidos pelo iPhone, sendo alguns deles a interface multitoque, e a utilização das redes 3G e *wireless*. Por fim, espera-se que esta ferramenta auxilie no trabalho dos profissionais da área médica agilizando a visualização das imagens radiológicas, para que assim possibilite um melhor relacionamento entre médico e paciente.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma ferramenta que possibilite a visualização das imagens radiológicas no formato JPEG para o dispositivo móvel iPhone.

Os objetivos específicos do trabalho são:

a) visualizar imagens radiológicas 2D no formato JPEG no iPhone;
b) utilizar os recursos de interação do iPhone, para prover as funcionalidades de zoom, ajuste de brilho, contraste e marcação de regiões da imagem;
c) visualizar as imagens radiológicas 3D a partir de um servidor utilizando a tecnologia 3G e/ou *wireless*.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em quatro capítulos.

O capítulo dois contém a fundamentação teórica necessária para fornecer um melhor entendimento sobre o tema proposto.

O capítulo três apresenta o desenvolvimento do visualizador de imagens, onde são explanados os principais requisitos do problema e a especificação contendo diagramas de classes e os casos de uso. Neste são apresentadas as ferramentas utilizadas, na implementação, no desenvolvimento e por fim são comentados os resultados e discussão.

No quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são apresentados os recursos do iPhone, juntamente com sua linguagem de programação, o Objective-C. Na seção 2.2 é apresentado o formato das imagens radiológicas DICOM. Na seção 2.3 é feita uma explanação sobre os visualizadores de imagens. A seção 2.4, apresenta alguns trabalhos correlatos ao trabalho proposto. Por fim, a seção 2.5 apresenta as principais *frameworks* para o iPhone.

2.1 IPHONE E SUA LINGUAGEM DE PROGRAMAÇÃO

O iPhone, em seu lançamento no ano de 2007 desenvolvido pela Apple, causou um grande espanto, trazendo apenas um botão e a maioria dos seus recursos utilizando-se da tecnologia multitoque, assim permitindo que se interaja com a tela apenas com o toque dos dedos na sua superfície (SIQUEIRA 2007, p. 54).

Segundo Siqueira (2007, p. 56), "A APPLE dá uma aula de inovação ao lançar um telefone celular com acesso à internet e a música digital. A criação do iPhone produziu 200 novas patentes para a empresa. Isso é o dobro que o Brasil registrou num ano", demonstrando assim um grande apego à pesquisa que a empresa possui. Outro aspecto que se destaca no iPhone, além de seus recursos, é a seção de venda de aplicativos, chamada de *AppStore*, onde os clientes que possuem iPhone podem comprar e publicar aplicativos desenvolvidos por eles.

O desenvolvimento para iPhone ocorre a partir de uma biblioteca chamada iPhone SDK, distribuída gratuitamente pela Apple, e utiliza a linguagem Objective-C. A linguagem foi criada por Brad Cox e sua empresa, a StepStone Corporation sendo caracterizada, segundo Apple (2010a), como "um pequeno, mas poderoso conjunto de extensões ao padrão de linguagem ANSI C", baseada no Smalltalk.

A linguagem Objetive-C possui um conjunto de adições à linguagem de programação C, dando suporte a orientação a objetos. A linguagem suporta polimorfismo e é possível adicionar classes em tempo de execução. A Objective-C foi projetada para dar a linguagem C capacidade plena de programação orientada a objetos e fazer programas simples e diretos. Podendo ser comparada ao C++ e ao JAVA, a Objetive-C possui sua própria *Integrated Development Environment* (IDE), o Xcode, sendo que a programação para iPhone torna-se

muito restrita à plataforma do ambiente Mac OS X.

Segundo Mark e Lamarche (2009, p. 2), para poder publicar aplicativos na AppStore, é necessário se associar em um programa de desenvolvimento, que é dividido em duas categorias o *standard program e o enterprise program*.

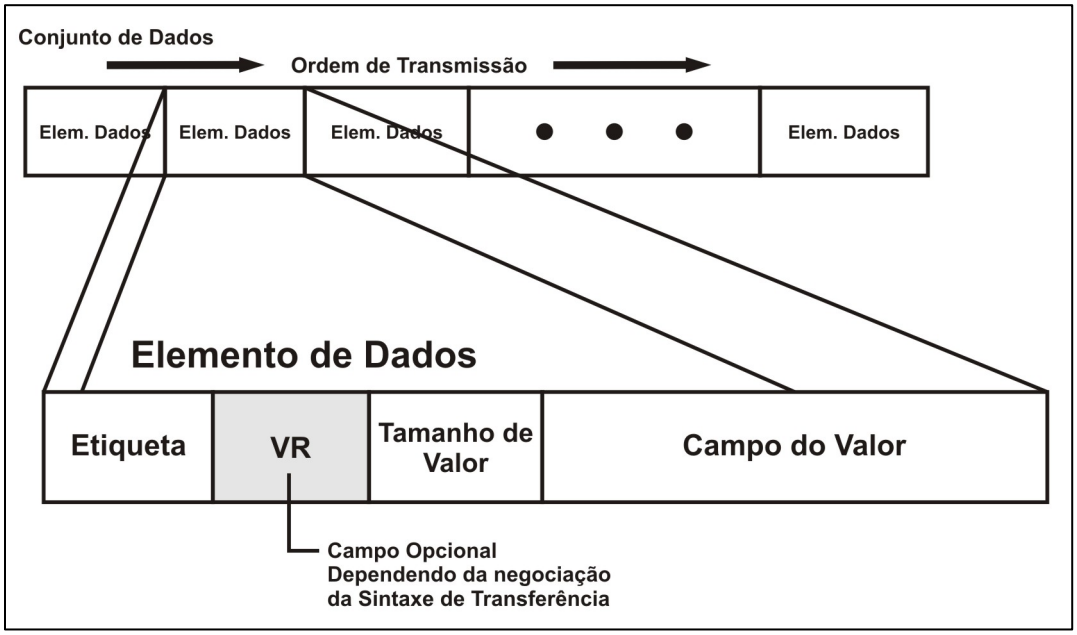
2.2 IMAGENS RADIOLÓGICAS

Segundo Stimac (1992, p. 3-4), na tomografia computadorizada uma ampola de raios X produz um feixe em forma de leque que atravessa uma seção (corte) do paciente. Este feixe em forma de leque é recebido por um arranjo circular de detectores do lado oposto do feixe, um computador emprega técnicas de reconstrução matemáticas para calcular um valor dentro de uma escala de tons cinza para cada pixel do corte. Com esta informação é gerada uma imagem eletrônica ou radiográfica.

Conforme Nema (2009), em 1985 o *American College of Radiologists* (ACR) e o NEMA reconheceram que era necessário definir um padrão de imagem que permitissem a conexão de aparelhos radiológicos de diferentes fabricantes. Sendo assim, criou-se o primeiro padrão, a versão 1.0 no *ACR/NEMA Standards Publication* no. 300-1985, modificada em 1988 para a versão 2.0, e em 1992-1993 com a *ACR/NEMA Standards Publication PS3*, conhecido como DICOM 3, sendo o padrão atual.

Somente em 1994 foi estabelecido o *Media Storage & File Format*, parte de número 10 do protocolo, que entre outras coisas define o formato de arquivo genérico DICOM, que contém um pequeno cabeçalho, o *Dicom File Meta Information Header*. Este com 128 bytes, que o usuário pode colocar qualquer informação, seguido do prefixo de 4 bytes e a extensão de arquivo *DICM*. Sendo seguido de uma mensagem DICOM que contém os elementos definidos no *File Meta Information* (grupo 0002) em uma sintaxe de transferência definida, que identifica unicamente o conjunto de dados, bem como a sintaxe do resto dos dados (NEMA, 2009).

Segundo Silva (2002), o conjunto de dados que formam o DICOM tem na composição do elemento quatro campos, sendo eles a etiqueta (*tag*); representação do valor do inglês *Value Represent* (VR); tamanho do valor e campo de dados. Os elementos de dados são especificados unicamente por uma etiqueta, ocorrendo a ordenação dos dados apenas por esta etiqueta, conforme mostrado na Figura 1.

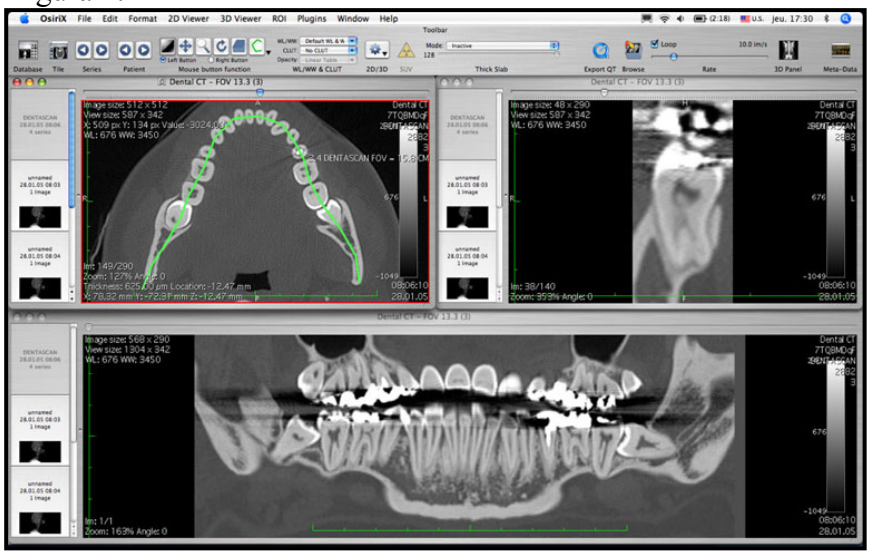


Fonte: Silva (2002).

Figura 1 - Visualização do elemento de dados DICOM

No DICOM 3, versão 3 da definição do formato, existem sintaxes compatíveis com as correspondentes no *Joint Photographic Experts Group* (JPEG), que têm compressões reversíveis, tirando proveito dos padrões de cores da imagem, e irreversíveis, descartando dados que o olho humano não reconhece facilmente. Uma imagem DICOM permite que se efetue *zoom* em pontos da imagem e proporcionando outras visões da imagem, conforme mostrado na

Figura 2.



Fonte: Osirix (2009).

Figura 2 – Visualização de uma imagem DICOM em vários ângulos

2.3 VISUALIZAÇÃO DE IMAGENS - JPEG

A computação gráfica, segundo Gomes e Velho (2003, p. 1), é o "Conjunto de métodos e técnicas para transportar dados em imagem através de um dispositivo gráfico". Definir esta área por muitas vezes é difícil, por que um de seus objetivos é o de transformar dados em imagens.

A computação gráfica por muitas vezes permite que seja possível visualizar objetos que fogem da realidade tridimensional. Conforme Gomes e Velho (2003, p. 4), "As operações de visualização consistem em uma sucessão de mudanças de sistemas de coordenadas entre os diversos espaços". Estas mudanças dão ao utilizador a sensação de 3D. Segundo Conci e Azevedo (2003, p. 11), "Há três categorias de estímulos visuais pelo cérebro para formar uma imagem 3D: informações monoculares, informações óculo-motoras e informações estereoscópicas".

As informações monoculares são inerentes a imagem formada na retina. Entre elas pode-se citar a perspectiva linear. As informações óculo-motoras são as fornecidas pelo movimento dos olhos, sendo divididas em duas categorias: a acomodação, onde os olhos relaxam ou contraem para mudar o formato do cristalino e a convergência, onde é considerado o grau de rotação dos olhos para obter informações a respeito da posição. As informações estereoscópicas são as informações obtidas pela posição de lugares diferentes dos olhos. Esta diferença é chamada de disparidade binocular.

As transformações nos objetos 3D que podem ser realizadas são (CONCI; AZEVEDO 2003, p. 12-14): translação, que significa movimentar o objeto; escala, que significa modificar as dimensões de escala; rotação, que significa girar a imagem; reflexão, que significa a visualização do próprio objeto como se fosse visualizado em um espelho; e cisalhamento, que significa uma transformação que distorce o formato do objeto.

Segundo Adams (1995, p. 55), "O uso eficaz da modelagem 3D e as técnicas de 'renderização' podem acrescentar realismo e exaltação ao seu software, especialmente se ele for um aplicativo que suporta animação, visualização ou simulação". A renderização refere-se aos cálculos matemáticos usados para resultar nos valores de iluminação para as entidades 3D.

Os objetos 3D muitas vezes são tratados como vários objetos 2D, pois facilitam o processo de aquisição e renderização. No caso de imagens matriciais, os objetos 2D podem ser representados por arquivos JPEG. Para se efetuar a visualização de imagens 3D é

necessário renderizar os objetos na área de visualização. Outra forma de se obter uma imagem 3D é utilizando várias visões de imagens 2D, como JPEG.

O JPEG é um método de compressão de imagens segundo, Paula Filho (2000, p.73), possuindo dois tipos de compressão, uma sem perda e outra com perda de qualidade. Segundo Paula Filho (2000, p.74) as técnicas de compressão sem perdas mantêm toda a informação da imagem original. A técnica que mais é usada apontada por Paula Filho (2000, p.74), é a de coerência de linhas, que verifica o número de linhas constantes da imagem e guarda o valor da cor e o comprimento da tira. Esta técnica é utilizada nas imagens *Graphics Interchange Format* (GIF).

Por sua vez na compressão com perda, são utilizadas técnicas onde existam perda de informações consideradas toleráveis, ou seja, onde a visão humana geralmente não consegue perceber a perda de qualidade da imagem. Um algoritmo central para este tipo de compressão, é o que transforma a imagem em um espectro[1], onde cada quadro a ser comprimido é dividido em blocos. O JPEG emerge atualmente como a técnica mais importante de compressão de imagens (PAULA FILHO 2000, p.74).

O padrão JPEG de uma forma simplificada envolve as seguintes etapas: obtenção do espectro bidimensional da imagem, truncamento dos campos do espectro para diminuir os dígitos menos significativos (desprezando os coeficientes próximos a zero) e a codificação entrópica dos componentes (semelhante à técnica usada pelos compressões de arquivos de fins gerais) (PAULA FILHO 2000, p.74).

2.4 TRABALHOS CORRELATOS

Existem alguns projetos que desempenham papel semelhante ao proposto neste trabalho. Dentre eles, foram escolhidos os trabalhos: “Conversão de Imagens do Formato DICOM Visando a Inter-Operacionalidade de Sistemas Através da WEB” (GUIMARÃES, 2002), “Análise de Imagens Radiológicas Via Dispositivos Móveis” (CORREA et al., 2007) e o software de visualização de imagens DICOM para Mac OS X “OsiriX” (OSIRIX, 2009) .

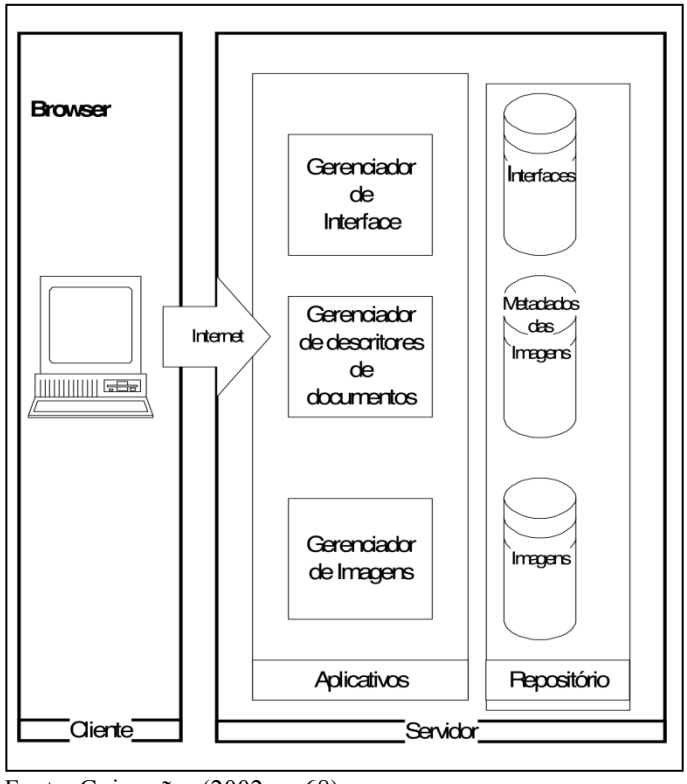
[1] Conjunto dos raios coloridos, resultantes da decomposição de uma luz complexa.

2.4.1 CONVERSÃO DE IMAGENS DO FORMATO DICOM VISANDO A INTER-OPERACIONALIDADE DE SISTEMAS ATRAVÉS DA WEB

O trabalho de conversão de imagens DICOM, realizado por Guimarães (2002), tem como objetivo permitir visualizar as imagens radiológicas num *browser*, onde foram utilizadas as seguintes tecnologias:

a) *Picture Archiving and Communication Systems* (PACS): responsável pelo transporte e armazenamento de imagens;
b) DICOM: é o padrão de imagens biomédicas;
c) *Health Level Seven* (HL7): é o padrão aprovado pela *American National Standards Institute* (ANSI) para a comunicação das imagens radiológicas;
d) *Common Object Request Broker Architecture* (CORBA): é utilizada para implantar as funcionalidades de comunicação de dados e suportar o grande volume de dados;
e) *Standard Generalized Markup Language* (SGML): é utilizado para representar o documento web.

O sistema utiliza a tecnologia envolvendo o conceito cliente/servidor e sua estrutura é descrita na Figura 3.



Fonte: Guimarães (2002, p. 68).

Figura 3 – Estrutura do sistema

Os resultados obtidos através deste trabalho demonstram que não é possível incorporar

todos os aspectos dos metadados no projeto, fazendo uso das funcionalidades oferecidas pelo padrão de transferência de imagens do padrão DICOM. Para isso foi necessário desenvolver uma interface que possibilitou a utilização de todos os aspectos deste padrão. Ainda, o sistema web não atende a flexibilidade exigida para a sua utilização no prontuário de pacientes.

2.4.2 ANÁLISE DE IMAGENS RADIOLÓGICAS VIA DISPOSITIVOS MÓVEIS

O trabalho de Correa et al. (2007) demonstra um sistema distribuído de análises de imagens médicas para a web, utilizando *web service*. É mantido um servidor que suporta diversas requisições de conversão de imagem. Segundo Correa et al. (2007), "A aplicação específica permite o recebimento do conjunto de imagens, ou parte dela, em uma única transmissão, permitindo a navegação pela pilha de imagens mesmo sem conectividade posterior".

Este sistema prevê uma análise prévia de uma imagem radiológica, o que facilita em caso de urgência ou rotina hospitalar. O sistema hoje já implementa algumas funcionalidades no *web service*, como a conversão das imagens de DICOM para JPEG e duas aplicações clientes, uma para *Personal Digital Assistants* (PDA) e outra para um telefone celular.

Neste trabalho foi verificado que nem sempre é necessário apenas visualizar as imagens para auxiliar um médico, pois muitas vezes espera-se que a ferramenta possa auxiliar na análise das imagens radiológicas.

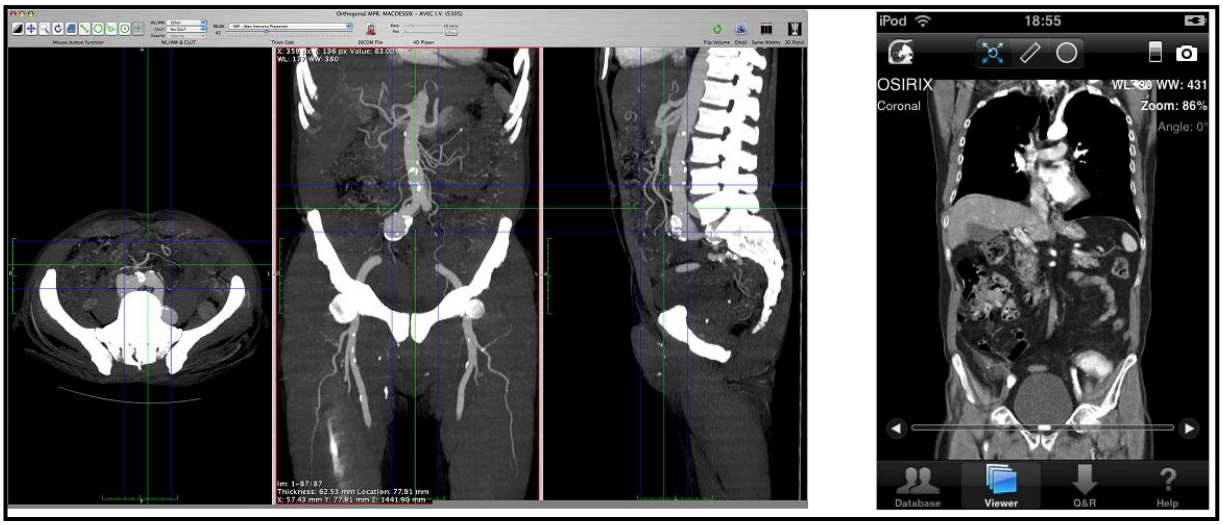
2.4.3 OSIRIX

O software OsiriX, segundo Osirix (2009), é dedicado para processar imagens DICOM produzidas por equipamentos de imagens, como Ressonância Magnética (RM), Tomografia Computadorizada (TC) e ultra-som. O OsiriX foi especificamente projetado para a navegação e visualização de imagens de multimodalidade e multidimensionais, sendo possível visualizar em: 2D, 3D, 4D e 5D.

Na visualização 4D tem-se uma imagem em 3D adicionando uma dimensão temporal. Já na imagem em 5D a imagem em 3D adiciona uma dimensão temporal e outra funcional. O software está disponível nos formatos 32-bits e 64-bits. A versão 64 bits permite carregar um número ilimitado de imagens, superior a 4 GB (limite na versão 32-bit). A versão 64 bits foi

totalmente otimizada para processadores Intel com multi-núcleos, oferecendo melhor desempenho na renderização 3D.

O projeto OsiriX começou em 2003 na Califórnia, Estados Unidos, concebido e desenvolvido pelo Dr. Antoine Rosset, com a ajuda de Joris Heuberger (um cientista da computação). O projeto OsiriX[2] disponibiliza o software para Mac OS X e a partir de novembro de 2008 conta com uma versão para o iPhone, com as mesmas funcionalidades de visualização que a versão para Mac OS X, permitindo o acesso as imagens por *wireless*, assim trazendo consigo uma nova perspectiva para a medicina, com a mobilidade na visualização das imagens. Na Figura 4 encontra-se um exemplo do software para Mac OS X e para iPhone.



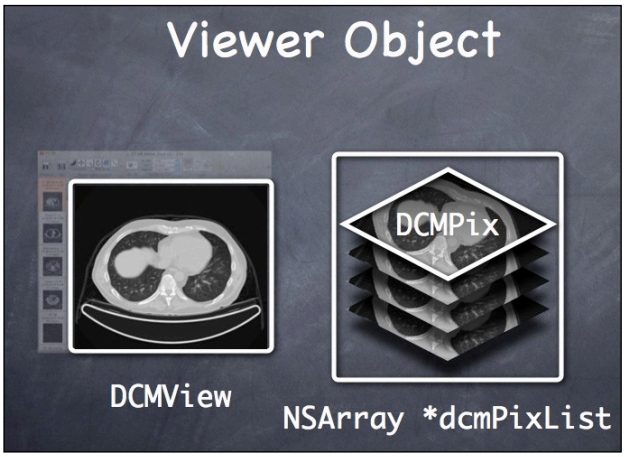
Fonte: Osirix (2009).

Figura 4 – Visualização de uma imagem a partir do Mac OS X e no dispositivo iPhone

O projeto Osirix para Mac OS X utiliza de outros projetos para *renderizar* as imagens, um deles é o Papyrus, que é uma biblioteca de grafos de cena em C++, que por sua vez utiliza a biblioteca Cairo para realizar as operações de desenho.

Segundo Osirix (2009), para efetuar a *renderização* das imagens, cada arquivo DCM (formato do padrão DICOM) é representada em um array de DCMPix, onde são interpretados para a exibição no DCMView, conforme Figura 5.

[2] O software é distribuído utilizando a licença *General Public License* (GPL).



Fonte: Osirix (2009).

Figura 5 – Renderização de uma imagem DCM no Osirix

2.5 FRAMEWORKS

Uma *framework* segundo Apple (2010c) é um pacote (um diretório estruturado) que contém uma biblioteca dinâmica compartilhada com rotinas que podem ser usadas em um programa. O acesso a esta biblioteca faz-se através do `header` da biblioteca. A seguir serão descritas algumas *frameworks* mais utilizadas no desenvolvimento de aplicações para o iPhone.

2.5.1 FOUNDATION

A *framework* Foundation define uma camada de base de classes Objective-C. Além de fornecer um conjunto de classes de objetos primitivos, ele apresenta vários paradigmas que definem funcionalidades não abrangidas pela linguagem Objective-C (APPLE 2010d, p. 26).

A estrutura da *framework* é projetada com esses objetivos em mente:

a) fornecer um conjunto pequeno de classes básicas;
b) permitir um desenvolvimento de software fácil, através da introdução de convenções consistentes para coisas como desalocação;
c) suporte a `strings Unicode`, a persistência do objeto, e objeto de distribuição;
d) proporcionar um nível de independência do sistema operacional, para aumentar a portabilidade.

Segundo Apple (2010d, p. 27), a framework Foundation inclui a classe de objetos raiz, classes que representam tipos de dados básicos, tais como `strings` e `arrays de bytes`, classes de coleção para armazenar outros objetos, classes que representam as informações do sistema, tais como datas, e as classes que representam as portas de comunicação. A identificação de objetos no código é facilitada pelo fato de todos as classes da *framework* Foundation iniciarem com "NS".

Por exemplo, a classe `NSNumber`, é uma subclasse de `NSValue` que oferece um valor como qualquer tipo escalar numérico em C (APPLE 2010d, p. 749). Ela define um conjunto de métodos especificamente para a definição e acesso de valores com um sinal ou não (`char`, `short int`, `int`, `long int`, `long long int`, `float` ou `double`).

Um outro exemplo segundo APPLE (2010d, p. 615), é a classe `NSMutableArray`, que declara a interface de programação de objetos para gerir um array de objetos modificáveis. Esta classe acrescenta inserção e as operações de exclusão para a matriz básica de manipulação de comportamento herdados do `NSArray`.

2.5.2 UIKIT

A *framework* UIKit fornece as classes necessárias para construir e gerir uma interface de usuário do aplicativo para o iPhone e iPod Touch. Com esta *framework* se faz possível a criação de eventos, desenhos, janelas e controles, todos projetados para suporte a multitoque (APPLE, 2010f, p. 24).

Segundo Apple (2010f, p. 25), todos os objetos da classe UIKit são herdados da classe `NSObject`. Como a *framework* Foundation, que iniciam com "NS", as classes da framework UIKit iniciam com "UI".

A classe `UIImage` é uma forma de alto nível para exibir uma imagem. Pode-se criar imagens a partir de objetos de imagem `Quartz`, ou a partir de dados brutos recebidos (APPLE, 2010f, p. 299). Segundo Apple (2010f), estes objetos da classe `UIImage` são imutáveis, ou seja, os dados não poderão ser alterados apos sua criação.

A classe `UIImageView` é uma classe que fornece uma container para exibir uma única imagem ou para animar uma série de imagens. Para animar as imagens, a classe `UIImageView` oferece controles para ajustar a duração e a freqüência da animação. Pode-se também iniciar e parar a animação livremente (APPLE, 2010f, p. 325).

Segundo Apple (2010f, p. 173), a classe `UIButton`, é uma classe que fornece os botões multitoque para a interface. O botão intercepta o evento de toque e envia uma mensagem ao objeto alvo.

A classe `UISlider` é a classe que é um controle visual usado para selecionar um único valor de uma faixa contínua de valores.

No Anexo A é apresentada a hierarquia das classes do *framework* UIKit.

2.5.3 QUARTZ CORE

A framework Quartz, segundo Apple (2010e, p. 7), fornece a API que suporta processamento de imagens e manipulação de imagem de vídeo. A identificação de objetos no código é facilitada pelo fato de todos as classes da framework Quartz Core iniciarem com "CA".

Por exemplo a classe `CATransition` segundo Apple (2010e, p. 119-123), implementa as animações de transição para uma camada. Pode-se também especificar os efeitos de transição de um conjunto de transições predefinidas. Alguns tipos de transição são:

a) `kCATransitionFade`: é o efeito do conteúdo da camada desaparecer à medida que se torna visível ou oculto;
b) `kCATransitionMoveIn`: é o efeito do conteúdo da camada se mover sobre a outra;
c) `kCATransitionPush`: é o efeito de empurrar uma imagem sobre qualquer conteúdo existente;
d) `kCATransitionReveal`: é o efeito do conteúdo da camada ser revelada gradualmente na direção especificada pelo subtipo de transição, as direções permitidas pelos subtipos são, esquerda, direita, de baixo e superior.

Ou ainda segundo Apple (2010e, p. 85-87) a classe `CAMediaTimingFunction` representa um segmento de uma função que define o ritmo de uma animação como uma curva de tempo. A função mapeia um tempo de entrada normalizados para o intervalo [0,1] a um tempo de saída também no intervalo [0,1].

2.5.4 CORE GRAPHICS

A framework Core Graphics segundo Apple (2010b, p. 9), é uma API baseada em C que foi influenciada no motor avançado de desenho Quartz. Ele permite com uma baixo custo, a renderização de objetos 2D. Esta estrutura pode ser usada para manipular com desenhos baseados em caminho, transformações de gerenciamento de cores, *renderização de offscreen*, padrões, gradientes e sombras, gerenciamento de dados de imagem, criação de imagem, máscaras, criação de documentos PDF, visualização e análise. As classes definidas neta framework iniciam sempre com "CG".

Para exemplificar a classe `CGContextRef` é uma região para se desenhar uma imagem 2D. A framework Core Graphics fornece funções de criação de vários tipos de regiões de desenho, incluindo imagens bitmap e PDF (APPLE, 2010b, p. 47).

Outro exemplo seria a classe `CGImageRef`, que segundo Apple (2010b, p. 210) serve para encapsular as informações de uma imagem bitmap. Para se criar um contexto uma das formas é utilizar um dos seguintes método `CGBitmapContextCreate` ou `ContextCreateImage`. Já para se copiar dados de um contexto existente é utilizado o método `CGDataProviderCopyData`.

3 DESENVOLVIMENTO

Neste capítulo serão abordadas as etapas de desenvolvimento do projeto. Onde estão descritos o projeto Osirix, os principais requisitos, a especificação, a implementação e por fim descrito os resultados e discussão.

3.1 PROJETO OSIRIX

Os passos necessários para obter e executar o código fonte do projeto Osirix são:

a) sistema operacional Mac OS X 10.5 Leopard (10.6 ou 10.7 recomendado);
b) um versionador de versões Subversion;
c) o IDE Xcode 3.2.

Já o procedimento para compilar e executar o projeto Osirix é:

a) abra uma aplicação Terminal (`localizado /Applications/Utilities/`);
b) na console digite o seguinte comando para baixar o código fonte:
   `svn co https://osirix.svn.sourceforge.net/svnroot/osirix osirix`;
c) abra o arquivo `Osirix.xcodeproj` localizado na pasta `osirix`;
d) acesse as `Targets` e clicando com o botão direito sobre a `Target`, clique em `Unzip Binaries`, escolha a opção `Build Unzip Binaries`;
e) se ocorrer algum erro no canto inferior direto da tela conforme a Figura 6, encerre o `Build` e execute o próximo passo;
f) clique sobre a exclamação e será apontado onde está o ocorrendo o erro, clique com o botão direito sobre o erro e selecione a opção `Open These Lastet Results as Transcript Text File`;
g) selecione e copie a linha de execução onde ocorreu o erro de compilação e a execute no terminal conforme a Figura 7 e a Figura 8.
h) após isso acesse o Xcode e execute o `Build and Run`, caso ocorra o problema de compilação execute novamente esse procedimento.

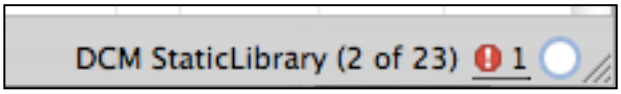


Figura 6 - Compilação com erros

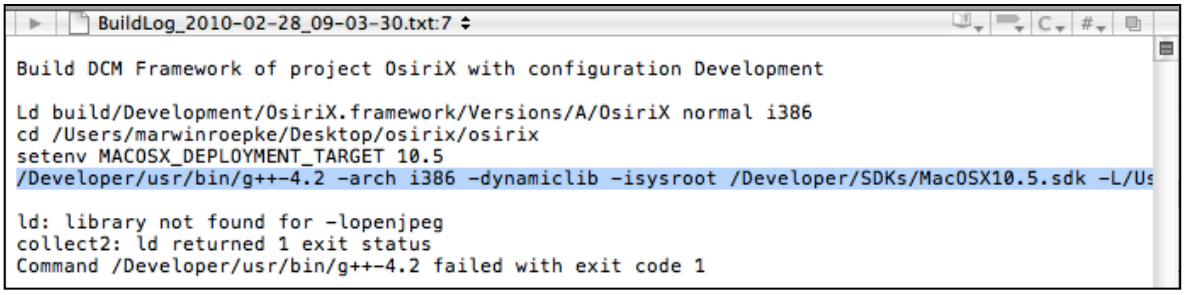


Figura 7 – Opção `Open These Lastet Results as Transcript Text File`

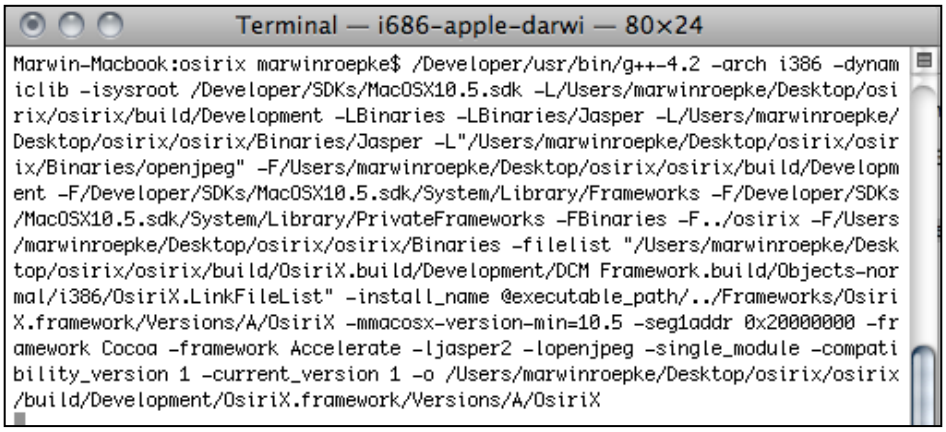


Figura 8 - Execução manual

Com a reinstalação do XCode foi observado que não se fez necessário executar este procedimento. Visto que as versões de softwares foram mantidas as mesmas.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos apresentados abaixo encontram-se classificados em Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF).

O visualizador de imagens radiológicas deverá atender aos seguintes requisitos:

a) o sistema deverá utilizar o multitoque para dar *zoom* as fotos (RF);
b) o sistema deverá permitir adicionar brilho e contraste as imagens (RF);
c) o sistema deverá permitir inserir marcações na imagem (RF);
d) o sistema deverá utilizar o acelerômetro para retirar as marcações na tela (RF);
e) o sistema deverá acessar imagens utilizando a rede 3G e/ou *wireless* (RNF);
f) o sistema deverá ser implementado na linguagem Objective-C (RNF);

g) o sistema deverá utilizar o ambiente Xcode para ser desenvolvido (RNF).

3.3 ESPECIFICAÇÃO

A especificacão do presente trabalho foi desenvolvida utilizando alguns dos diagramas da Unified Modeling Language (UML) em conjunto com a ferramenta Enterprise Architect 6.5.806 para a elaboração dos casos de uso e dos diagramas de classes.

3.3.1 Diagrama de casos de uso

Nesta seção são descritos os casos de uso para que o usuário possa ter uso de todos os recursos disponibilizados pelo visualizador. A Figura 9 apresenta os diagrama de casos de uso do visualizador.

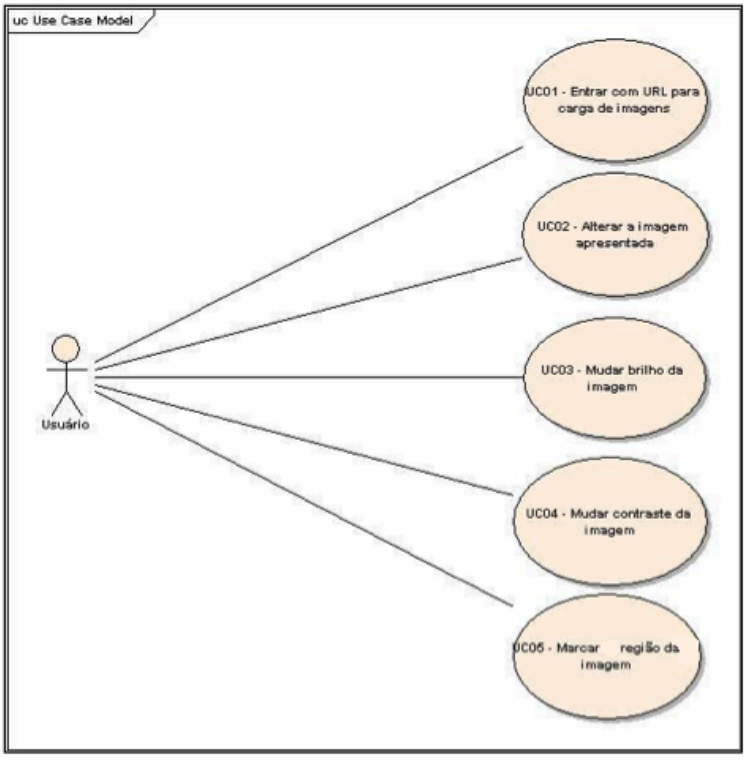


Figura 9 – Diagrama de casos de uso

3.3.1.1 UC01 – Entrar com URL para carga de imagens

O caso de uso UC01 - Entrar com URL para carga de imagens (Quadro 1) descreve como o usuário informa para o visualizador o local em que se encontram as imagens a serem carregadas.

UC01 – Entrar com URL para carga de imagens: possibilita ao usuário informar o local que encontra-se as imagens.	
Pré-Condição	1. As imagens estarem no formato JPEG, disponibilizados para acesso pelo protocolo HTTP. 2. Possuir o arquivo de índice chamado Arquivo.txt.
Cenário Principal	1. O usuário informa a url no campo URL apresentado na tela. 2. O usuário clique no botão Done.
Exceção	No passo 1 se for informado uma URL inexistente, será exibida uma mensagem para informar uma nova URL.
Pós-Condição	A carga das imagens da URL fornecida são armazenadas em um NSMutableArray, para apresentar ao usuário.

Quadro 1 – Caso de uso UC01

3.3.1.2 UC02 – Alterar imagem apresentada

O caso de uso UC02 – Alterar imagem apresentada (Quadro 2) descreve as formas com que o usuário pode visualizar uma outra imagem carregada.

UC02 – Alterar imagem apresentada: possibilita ao usuário altere a imagem que se esta visualizando.	
Pré-Condição	1. O usuário deve ter executado o caso de uso UC01. 2. Estar com na opção Animation.
Cenário Principal	1. O usuário clica no botão +, para avançar a imagem apresentada, ou no botão – para retornar a uma imagem anterior.
Cenário Alternativo	1. O usuário arrasta o componente UISlider para a direita para avançar a imagem apresentada ou para esquerda para retornar a uma imagem anterior.
Cenário Alternativo	1. O usuário clica no botão Auto para iniciar a transição de imagens para a próxima imagem.
Pós-Condição	A imagem apresentada é alterada para a anterior ou próxima.

Quadro 2 – Caso de uso UC02

3.3.1.3 UC03 – Mudar brilho da imagem

O caso de uso UC03 – Mudar brilho da imagem (Quadro 3) descreve a forma com que o usuário pode aumentar e diminuir o brilho na imagem.

UC03 – Mudar brilho da imagem: possibilita ao usuário altere a quantidade de brilho na imagem que se esta visualizando.	
Pré-Condição	1. O usuário deve ter executado o caso de uso UC01. 2. Estar na com a tab Brightness selecionada.
Cenário Principal	1. O usuário arrasta o componente UISlider para a direita para aumetar a quantidade de brilho ou para esquerda para diminuir a quantidade de brilho.
Pós-Condição	A mudança de brilho da imagem apresentada.

Quadro 3 – Caso de uso UC03

3.3.1.4 UC04 – Mudar contraste da imagem

O caso de uso UC04 – Mudar contraste da imagem (Quadro 4) descreve a forma com que o usuário pode aumentar e diminuir o contraste na imagem.

UC04 – Mudar contraste da imagem: possibilita ao usuário altere a quantidade de contraste na imagem que se esta visualizando.	
Pré-Condição	1. O usuário deve ter executado o caso de uso UC01. 2. Estar com a tab Contrast selecionada.
Cenário Principal	1. O usuário arrasta o componente UISlider para a direita para aumentar a quantidade de contraste ou para esquerda para diminuir a quantidade de contraste.
Pós-Condição	A mudança de contraste da imagem apresentada.

Quadro 4 – Caso de uso UC04

3.3.1.5 UC05 – Marcar região da imagem

O caso de uso UC05 – Marcar região da imagem (Quadro 5) descreve a forma com que o usuário pode marcar uma região da imagem para uma análise posterior.

UC05 – Marcar região da imagem: possibilita ao usuário que efetue uma marcação na imagem apresentada na forma circular.	
Pré-Condição	1. O usuário deve ter executado o caso de uso UC01. 2. Estar com a tab Mark selecionada.
Cenário Principal	1. O usuário clica em um ponto da imagem e arrasta até outro ponto, para desenhar uma elipse.
Pós-Condição	Desenhar uma elipse entre estes dois pontos.

Quadro 5 – Caso de uso UC05

3.3.2 Diagrama de classes

Na Figura 10 é apresentado o diagrama de classes, exibindo as principais classes utilizadas neste projeto. Para melhor visualização, o diagrama de classes será tratado por cada classe em separado nas seções seguintes.

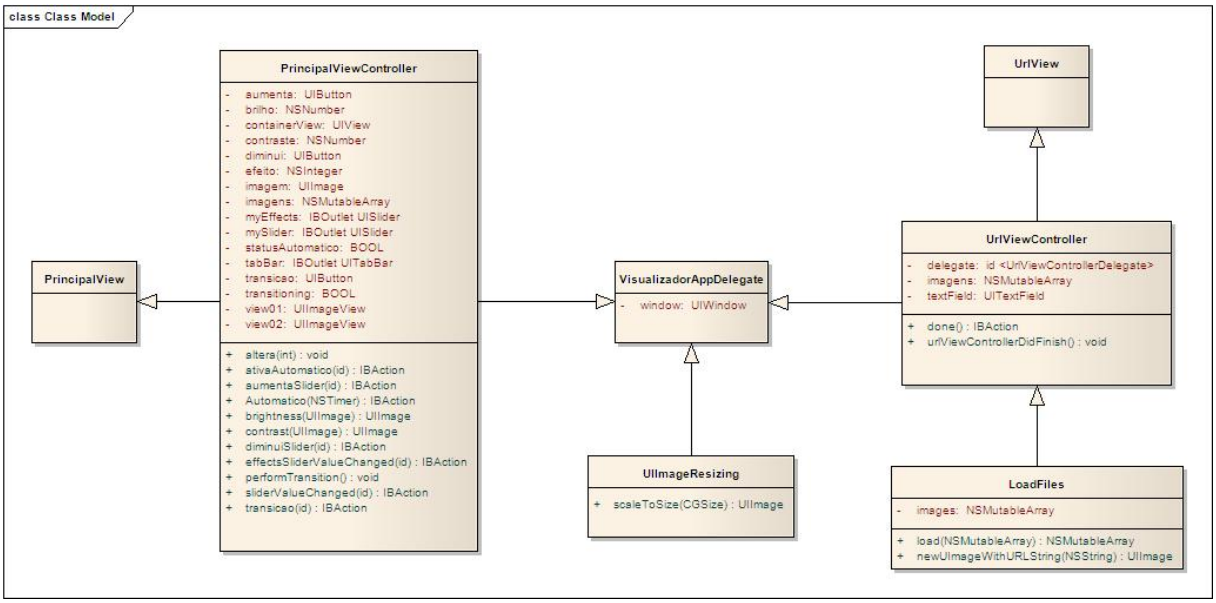


Figura 10 – Diagrama de classes do visualizador

3.3.2.1 Classe `PrincipalViewController`

A classe `PrincipalViewController` é a que realiza a manipulação das imagens, ela é uma classe que efetua desde a *renderização* das imagens, aos efeitos de brilho e contraste da imagem. A Figura 11 mostra o diagrama de classe referente a classe `PrincipalViewController`.

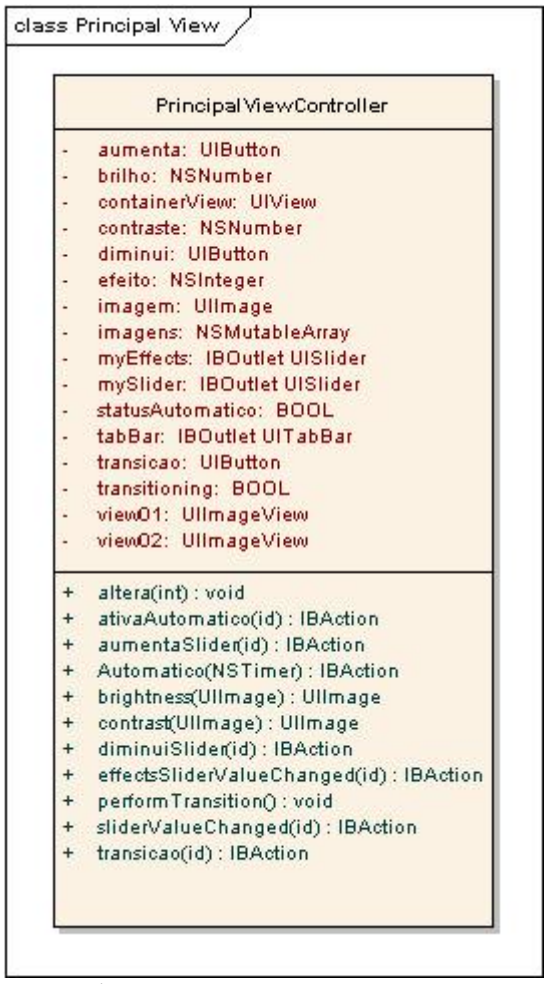


Figura 11 - Classe `PrincipalViewController`

O método `altera` é responsável pela alteração da imagens armazenada no `UIView`. Sendo que este pode ser acionado pela alteração do `slider`, pelo clique no botão automático e pelos botões `+` e `-`. Ele efetua a alteração da imagem que esta no `UIView02`. Após a alteração da imagem no `UIView02` é chamado o método `performTransition`, que efetua a transição da imagem com o efeito de animação.

3.3.2.2 Classe `UIImageResizing`

A classe `UIImageResizing` realiza o ajuste de escala da imagem para o tamanho da tela do iPhone. A Figura 12 mostra o diagrama de classe referente a classe `UIImageResizing`.

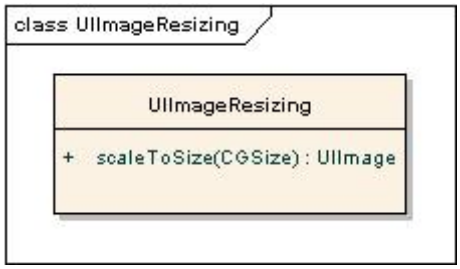


Figura 12 - Classe `UIImageResizing`

3.3.2.3 Classe `VisualizadorAppDelegate`

A classe `VisualizadorAppDelegate` conforme a Figura 13 é responsável por efetuar a transição entre as telas da aplicação.

Figura 13 - Classe `VisualizadorAppDelegate`

Ela possui dois métodos o `flipToBack` e `flipToFront`. O método `flipToBack` efetua a transição para a tela da classe `principalViewController`. Já o método efetua a transição para a tela da classe `flipToFront` efetua a transição para a tela da classe `urlViewController`.

3.3.2.4 Classe `UrlViewController`

A classe `UrlViewController` é a que fica responsável pela tela inicial do visualizador para efetuar a carga das imagens. A Figura 14 mostra o diagrama de classe referente a classe `UrlViewController`.

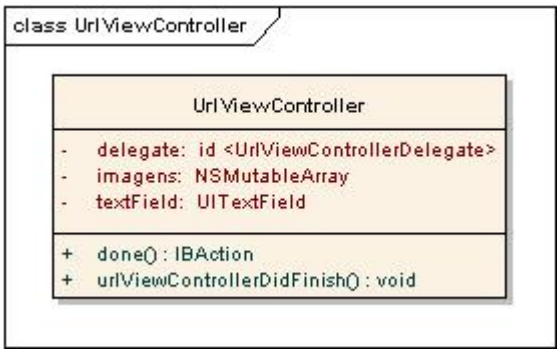


Figura 14 - Classe `UrlViewController`

3.3.2.5 Classe `LoadFiles`

A classe `LoadFiles` é a que efetua a carga das imagens de uma URL para uma `NSMutableArray`, usado para armazenamento local das imagens enquanto de sua execução. A Figura 15 mostra o diagrama de classe referente a classe `LoadFiles`.

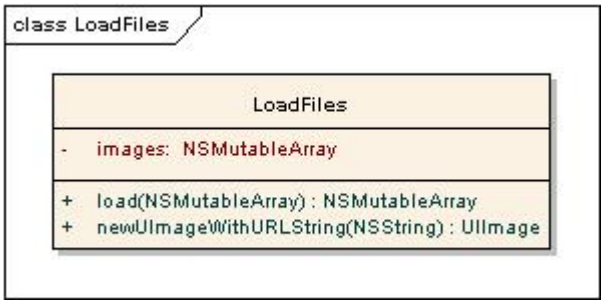


Figura 15 - Classe `LoadFiles`

O método `load` busca o arquivo de referencia na URL fornecida e instância um objeto de `NSMutableArray` que conterá todas imagens para o programa. Estas imagens são obtidas com a passagem da url completa da imagem para o método `newUIImageWithURLString` que retorna um objeto de `UIImage` do caminho fornecido.

3.4 IMPLEMENTAÇÃO

As técnicas e ferramentas utilizadas, bem como a operacionalidade do visualizador desenvolvido neste trabalho, são apresentadas a seguir. Serão apresentadas todas as ferramentas utilizadas e as principais funções que o visualizador possui, como ajuste de escala, carga das imagens, transição das imagens e os efeitos de brilho e contraste.

3.4.1 Técnicas e ferramentas utilizadas

As tecnologias utilizadas para o desenvolvimento foram as seguintes:

a) linguagem Objective-C;
b) a Integrated Development Environment (IDE) xCode 3.2.1(para codificação) e do Interface Builder versão 3.2.1 (para desenho de interfaces), disponibilizados pela Apple;
c) para executar o visualizador, foi utilizado o Iphone Simulator 3.1, que funciona integrado com a IDE do xCode;
d) as seguintes frameworks também fornecidas junto com o xCode `UIKit`, `Foundation`, `CoreGraphics` e `Quartz Core`.

3.4.2 Ajuste de escala das imagens

O ajuste das imagens para o iPhone ocorre conforme o Quadro 6. Na linha 15 é definido um contexto para uma imagem, que é usado para instanciar um objeto `CGContextRef` na linha 17, onde este representa um ambiente de desenho da biblioteca Quartz 2D. Na linha 18 ele altera a origem do sistema de coordenadas para um contexto, já na linha 19 ele altera a escala deste sistema.

Com o objeto instanciado na linha 21 é desenhada a imagem na escala passada no objeto instanciado. Por fim na linha 23 é atribuído a um objeto `UIImage`, a imagem montada no objeto da biblioteca Quartz 2D. Que no fim de sua execução é retornado para quem chamou o método.

```
14  - (UIImage*)scaleToSize:(CGSize)size {
15      UIGraphicsBeginImageContext(size);
16
17      CGContextRef context = UIGraphicsGetCurrentContext();
18      CGContextTranslateCTM(context, 0.0, size.height);
19      CGContextScaleCTM(context, 1.0, -1.0);
20
21      CGContextDrawImage(context, CGRectMake(0.0f, 0.0f, size.width, size.height), self.CGImage);
22
23      UIImage* scaledImage = UIGraphicsGetImageFromCurrentImageContext();
24
25      UIGraphicsEndImageContext();
26
27      return scaledImage;
28  }
```

Quadro 6 - Ajuste de escala das imagens

3.4.3 Carga das imagens a partir de uma URL

Para realizar a leitura das imagens para o projeto foi utilizado um arquivo com o nome de `Arquivos.txt` que contém a referência dos arquivos a serem carregados. Este arquivo possui os campos mapeados conforme Figura 16.

IM-0001- Padrão das imagens	
0002	Inicio da numeração
0352	Fim da numeração

Figura 16 – Arquivo de referência

A leitura de arquivo é efetuada pela classe `CarregaArquivos`, pelo método `(NSMutableArray*)carrega:(NSString*) url`. Ele recebe como parâmetro uma string com a URL onde será feita toda a leitura das imagens, conforme a linha 20 do Quadro 7.

```
15  -(NSMutableArray*)carrega:(NSString*) url{
16
17      self.imagens  = [[NSMutableArray alloc] init];
18      //Busca arquivo de referencia das imagens
19
20      NSString *caminhoImagens = [[NSString alloc] initWithString:[NSString stringWithFormat:@"%@%@",url,@"Arquivos.txt"]];
21
22      NSString *myText = [NSString stringWithContentsOfURL:[NSURL URLWithString:caminhoImagens]];
23      NSLog(@"Tamanho String: %d", [myText length]);
24
25      NSMutableString *resto;
26      NSRange range = [myText rangeOfString:@"\n"];
27      int indice = range.location;
28
29      NSString *padrao_arquivos = [[NSString alloc] initWithString:[myText substringToIndex:indice]];
30      resto = [myText substringFromIndex:indice+1];
31      range = [resto rangeOfString:@"\n"];
32      indice = range.location;
33
34      //inicia a marcação do primeiro e do ultimo arquivo
35      int inicio = [[[NSString alloc] initWithString:[resto substringToIndex:indice]] intValue];
36      int final    = [[[NSString alloc] initWithString:[resto substringFromIndex:indice+1]] intValue];
37
38      //Define quantidade de arquivos
39      int tamanho = (final - inicio);
40      //Adiciona no NSMutableArray o nome dos arquivos a serem carregados
41      NSString *nome_caminho = [[NSString alloc] initWithString:(@"")];
42      for (int cont = 0; cont <=tamanho; cont++) {
```

Quadro 7 - Carga dos arquivos para `NSMutableArray`

Na linha 17 é instanciado o objeto `NSMutableArray`, que armazena as imagens a serem carregadas. Nas linhas 20 a 36, o arquivo de referência é lido e alocado nas variáveis para serem usados no loop que inicia na linha 42 de carga dos arquivos.

Para se poder entrar com a URL de onde será feita a leitura o usuário clica sobre o ícone de informação depois de clicado sobre o ícone será apresentado a tela para informar a URL onde estão as imagens conforme a Figura 17.

Figura 17- Informar URL para leitura das imagens

3.4.4 Transição das imagens

A transição das imagens dando o efeito de animação é feito pela classe `VisualizadorView`, ela ocorre conforme o Quadro 8. Na linha 250, é configurado que a transição entre uma imagem e outra é de 1 segundo, para que tenha uma percepção de suavidade. Na linha 255 é configurado o tipo de transição para `kCATransitionFade`, onde o conteúdo da camada desaparece à medida que se torna visível ou oculto a outra.

```
245 -(void)performTransition
246 {
247     // Primeiro crie um objeto CATransition para descrever a transição
248     CATransition *transition = [CATransition animation];
249     // Anima por 1 segundo
250     transition.duration = 1;
251     // usando a função timing
252     transition.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionDefault];
253
254     // Seta o tipo de transiçao
255     transition.type = kCATransitionFade;
256
257     // Finalmente, para evitar a sobreposição transições nós atribuímos a nós mesmos a delegaçao para a animação e
        esperamos a
258     //-AnimationDidStop:finish:message. Quando ele vem, a flag já não esta em transição.
259     transitioning = YES;
260     transition.delegate = self;
261
262     // Em seguida adicioná-lo à camada de containerView. Isto irá executar a transição com base em como podemos alterar o
        conteúdo.
263     [containerView.layer addAnimation:transition forKey:nil];
264
265     // Aqui nós temos a view1 escondida e a View2 mostrada, o que fará com que o Core Animation para animar view1 um hora
        e outra a View2
266
267     view01.hidden = YES;
268     view02.hidden = NO;
269
270     // E assim que vamos continuar a trocar entre as duas imagens, trocamos as variáveis de instância.
271     UIImageView *tmp = view02;
272     view02 = view01;
273     view01 = tmp;
274 }
```

Quadro 8 - Transição de imagens

Na linha 259, é alterado o valor que permite efetuar a transição, na linha 260 configura-se a delegação da animação para o objeto de `CATransition`, para evitar que uma sobreposição de transições.

Esta transição pode ser feita de forma manual pelo usuário clicando nos botões `+`, para avançar e `-`, para retornar a imagem. A outra forma é clicando no botão `Auto`, como pode ser observado na Figura 18.

Figura 18 - Mudar imagem apresentada

3.4.5 Efeitos de brilho e contraste

Os efeitos de brilho e contraste são obtidos com a alteração dos dados do modelo de cores RGB (*Red, Green, Blue*) da imagem.

Para se obter o brilho é efetuado alteração da imagem com a amplitude das variações indo para a cor branca conforme Figura 19. Já no efeito de contraste é a amplitude das variações em níveis de cinza conforme Figura 20.

Figura 19 - Mudança de brilho

Figura 20 - Mudança de contraste

A alteração de brilho e contraste tem como base o mesmo método, que foi optado em buscar um pixel e alterar os valor do RGB de todos pixels. Na linha 85 do Quadro 9 é criado uma variável `inImage` que é do tipo `CGImageRef`, que é a representação de uma imagem bimap. Na linha 89 é declarada uma variável do tipo `CFDataRef`, que é utilizado para buffer de bytes, para armazenar os pixels da imagem. Na linha 95 é efetuado um laço `for` que efetua a alteração dos valores dos pixels.

```
85          CGImageRef inImage=imagem.CGImage;
86
87          CGContextRef ctx;
88
89          CFDataRef m_DataRef;
90          m_DataRef = CGDataProviderCopyData(CGImageGetDataProvider(inImage));
91          UInt8 * m_PixelBuf = (UInt8 *) CFDataGetBytePtr(m_DataRef);
92          int length = CFDataGetLength(m_DataRef);
93          CGImageGetBitsPerComponent(inImage), CGImageGetBitsPerPixel(inImage), CGImageGetBytesPerRow(inImage);
94
95          for (int index = 0; index < length; index += 4)
96          {
97              Byte tempR = m_PixelBuf[index + 1];
98              Byte tempG = m_PixelBuf[index + 2];
99              Byte tempB = m_PixelBuf[index + 3];
100
101             int outputRed = level + tempR;
102             int outputGreen = level + tempG;
103             int outputBlue = level + tempB;
104
105             if (outputRed>255) outputRed=255;
106             if (outputGreen>255) outputGreen=255;
107             if (outputBlue>255) outputBlue=255;
```

Quadro 9 - Alteração do brilho da imagem

3.4.6 Efeito de marcação nas imagens

A marcação na imagem, é obtido com o desenho de elipses sobre a imagem apresentada ao usuário. Para isto é utilizado os eventos `touchesBegan`, `touchesMoved` e `touchesEnded`, que retornam os eventos de toque na tela. Sendo que o `touchesBegan` é usado para guardar o primeiro toque na tela, o `touchesMoved` é usado para atualizar o ultimo toque na tela e o `touchesEnded` é usado para desenhar na tela.

Conforme o Quadro 10, é demonstrado como é desenhado na tela, na linha 65 efetua-se uma verificação se o usuário esta na opção `Mark`. Na linha 71 definimos o contexto com o tamanho da `UIView` atual. Já na linha 72, cria-se uma retângulo iniciando com as coordenadas do x, y iniciais e finais.

Com o contexto declarado e o retângulo onde se vai desenhar para usuário, configura-se a cor da linha para vermelho na linha 74 e desenha-se a elipse na linha 76. Esta elipse é desenhada passando o ponto de x, y que obtemos pelo evento `touchesBegan`, e a largura e altura, obtidas com o cálculo de pontos iniciais e finais obtidas pelos eventos de `touchesMoved`.

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    if (efeito ==4) {
        if ((firstTouch.x > 0) && (firstTouch.y > 0) && (lastTouch.x > 0) && (lastTouch.y >0)) {
            //Limpa a tela caso aja qualquer figura.
            int valor = (int)round(([mySlider value]*[imagens count])/100);
            view01.image = [[imagens objectAtIndex:valor] scaleToSize:CGSizeMake(320.0f,320.0f)];
            view01.image = [self brightness:[self contrast:view01.image]];
            UIGraphicsBeginImageContext(self.view.frame.size);
            [view01.image drawInRect:CGRectMake(0, 0, self.view.frame.size.width, self.view.frame.size.
                height)];
            //Seleciona a cor vermelha para a linha
            CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 255.0, 0.0, 0.0, 1.0);
            //Desenha a Elipse
            CGContextAddEllipseInRect(UIGraphicsGetCurrentContext(), CGRectMake(firstTouch.x, (firstTouch.y
                *100 / (320*100/460)), lastTouch.x - firstTouch.x, lastTouch.y - firstTouch.y));
            CGContextStrokePath(UIGraphicsGetCurrentContext());

            view01.image = UIGraphicsGetImageFromCurrentImageContext();
            UIGraphicsEndImageContext();

            [view01 removeFromSuperview];
            [containerView addSubview:view01];
        }
        firstTouch.x = -1;
        firstTouch.y = -1;
        lastTouch.x = -1;
        lastTouch.y = -1;
    }
}
```

Quadro 10 - Código de marcação de região

A marcação conforme a Figura 21 (região hachurada), realizada pelo usuário na imagem, é apagada da quando é efetuado alguma alteração do brilho, contraste ou altera-se a imagem.

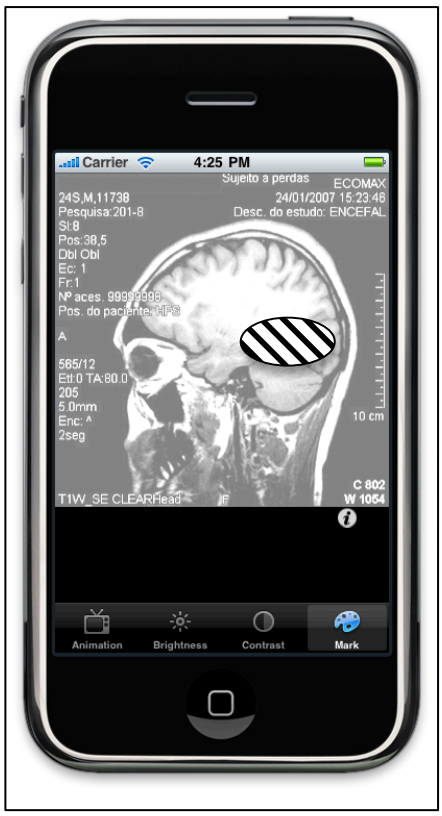


Figura 21 - Marcação de região da imagem

3.5 RESULTADOS E DISCUSSÕES

No inicio trabalhado-se na migração das rotinas de codificação de imagens DICOM utilizadas no projeto Osirix para Mac OS X, sendo que este utiliza tanto bibliotecas desenvolvidas por terceiros em C++, como bibliotecas disponibilizadas pela a própria Apple. Estas bibliotecas em sua maioria possuem uma biblioteca reduzida para o iPhone. No entanto teve-se dificuldades de migrar estas codificações para o iPhone.

Com esta dificuldade encontrada, optou-se em trabalhar com imagens JPEG, por ser um formato que o Osirix suporta exportar as imagens DICOM e ser um dos padrões de imagens suportados pelo iPhone. As imagens JPEG foram obtidas pelo programa Osirix, que permite exportar as imagens para JPEG. As imagens mantiveram 10 % de seu tamanho quando armazenadas no formato JPEG. O desenvolvimento da aplicação teve como base alguns exemplos disponíveis pela Apple, como os projetos GLImageProcessing, ViewTransitions e GLPaint.

O projeto GLImageProcessing segundo Apple (2010g), demonstra como implementar filtros de imagem simples de processamento (Brilho, Contraste, Saturação, Matriz rotação, Sharpness), utilizando OpenGL ES1.1. O exemplo também mostra como criar simples ícones processuais (definido matematicamente) usando CoreGraphics.

O projeto ViewTransitions segundo Apple (2010h), demonstra como executar as transições entre duas `UIView` usando transições do `Core Animation`. Ao olhar o código, você verifica como usar um objeto `CATransition` para criar transições e controle.

O projeto GLPaint segundo Apple (2010i), demonstra como o GLPaint apóia a pintura usando o touch com o OpenGL ES. Este exemplo também mostra como detectar um evento de movimento do dispositivo.

Com a aplicação desenvolvida foi efetuado o seguinte cenário de teste para que seja analisada a performance da aplicação. O Quadro 11 realiza-se uma relação de velocidade de carga das imagens com a velocidade da rede de dados, as imagens utilizadas tiveram 54 KB cada uma.

Quantidade de Imagens	Velocidade de Conexão	Tamanho Total	Tempo de Carga
25	1 MB	1,3 MB	29 segundos
50		2,6 MB	57 segundos
350		18,5 MB	6 minutos e 30 segundos
25	10 MB	1,3 MB	3 segundos
50		2,6 MB	5 segundos
350		18,5 MB	38 segundos
25	54 MB	1,3 MB	1 segundos
50		2,6 MB	1 segundos
350		18,5 MB	8 segundos

Quadro 11 - Relação tempo de carga VS velocidade conexão

Com a comparação verificou-se que a velocidade de conexão do iPhone foi um fator importante na para carga das imagens, pela espera na carga das imagens para ele. Pelo gráfico apresentado na Figura 22 pode-se ter uma outra visão sobre este tempo de carga das imagens, podendo levar o usuário a pensar que ocorreu um travamento da aplicação.

Figura 22 - Tempo de carga das imagens

Atualmente a visualização das imagens é feita com imagens no formato JPEG, o que pode impactar diretamente no aumento do acesso a estes imagens pela Web caso o usuário queira interagir com esta imagem mudando, por exemplo, a orientação da imagem 2D visualizada no momento. Ou ainda, se o usuário queira habilitar ou desabilitar o texto das informações do paciente, já que como o JPEG é uma matriz imagem impossibilita estas alterações. Nada impede que o servidor de imagens receba da aplicação cliente disponível no iPhone, solicitando imagens parametrizadas para que no servidor as gere em grupos de imagens ao "gosto do usuário", mas obviamente vai depender sempre das condições de

transmissão disponíveis.

Desta forma deve-se avaliar a relação de disponibilidade de transmissão de acesso versos recursos de processamento no dispositivo local (no caso o iPhone). Pois se tiver uma velocidade de acesso aceitável pode-se assumir que sempre serão transmitidas grupos de imagens JPEG de acordo com a solicitação do cliente (aplicação no iPhone) e todo o tratamento na imagem do tipo mudança de orientação e outras seriam feitas no servidor direto nas imagens DICOM.

Já se houver recursos de processamento locais (no iPhone) e não quiser ficar dependendo do acesso externo a Web, pode-se trabalhar para transmitir as imagens no formato DICOM e fazer o tratamento das imagens localmente, para depois gerar as imagens JPEG para serem visualizadas.

Um outro teste realizado foi de verificar a utilização de memória da aplicação. O teste consiste em abrir a aplicação e verificar o crescimento de utilização da memória quando da carga das imagens conforme o Quadro 12.

Quantidade de Imagens	Memória Real	Memória Virtual	Tamanho total das imagens	Memória Real	Memória Virtual
25	4,17 MB	53,53 MB	1,3 MB	9,45 MB	69,57 MB
50			2,6 MB	9,74 MB	69,74 MB
350			18,5 MB	15,68 MB	86,58 MB

Quadro 12 - Teste de utilização de memória

Este teste mostrou que partes das imagens são alocadas na região de memória real e outra parte na região de memória virtual.

4 CONCLUSÕES

Este trabalho permite que os usuários de iPhone possam ter a disponibilidade de uma forma mais prática e ágil a visualização de imagens radiológicas no formato JPEG. Podendo efetuar a transição das imagens, alterações de brilho e contraste, aumentar/diminuir o zoom e marcar regiões na imagem, trazendo uma maior comodidade para o paciente, e principalmente ao médico que tem um instrumento de auxílio ao prognóstico em suas mãos.

Já para a informática, o visualizador explorou a plataforma do desenvolvimento Objective-C para iPhone, sendo um dispositivo móvel com a integração de recursos como: acelerômetro, multitoque e transmissão Web, em uma tela de alta qualidade gráfica num dispositivo móvel.

O presente trabalho cumpriu o objetivo de demonstrar a possibilidade de acesso a imagens médicas a distância através de um dispositivo móvel, mesmo não sendo possível o acesso as imagens DICOM diretamente, mas sim usando imagens radiológicas no formato JPEG.

Este trabalho também conseguiu um outro importante resultado, o de permitir que o mesmo seja uma base para um futuro desenvolvimento de uma aplicação que permita a visualização de imagens médicas em 3D.

4.1 EXTENSÕES

Sugere-se, para futuros trabalhos, a utilização de imagens do formato DICOM para a visualização. Também sugere-se a utilização de persistência local das imagens, sendo efetuado apenas uma vez a carga das imagens ao dispositivo. Com a carga das imagens seja criado repositórios locais das imagens.

Para que seja amenizado o problema na demora na carga de um volume grande de imagens, seria aconselhável que a carga fosse feita em lotes de imagens e estas já fossem apresentadas ao usuário.

Recomenda-se também que seja possível enviar pela WEB ou por e-mail os prontuários com a marcação das áreas selecionadas.

Como melhoria pode-se desenvolver uma rotina de carga de imagens com uma

autenticação prévia, para garantir que apenas pessoas autorizadas tenham acesso as imagens das radiografias. Outra melhoria que pode ser implementada é a de uma forma de rotação das imagens apresentadas ao usuário.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, L. **Visualização e realidade virtual**: programação 3D com visual basic for windows. São Paulo: Makrom Books. 1995.

APPLE. **Introduction to the Objective-C 2.0 programming language**. Cupertino, 2010a. Disponível em: <http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html >. Acesso em: 21 mar. 2010.

_____. **Core graphics framework reference**, [S.l.], 2010b. Disponível em: <http://developer.apple.com/iphone/library/documentation/CoreGraphics/Reference/CoreGraphics_Framework/CoreGraphics_Framework.pdf>. Acesso em: 25 maio 2010.

_____. **Framework**. [S.l.], 2010c. Disponível em: <http://developer.apple.com/iphone/library/documentation/General/Conceptual/DevPedia-CocoaCore/Framework.html>. Acesso em: 25 maio 2010.

_____. **Foundation framework reference**. [S.l.], 2010d. Disponível em: <http://developer.apple.com/iphone/library/documentation/cocoa/Reference/Foundation/ObjC_classic/FoundationObjC.pdf>. Acesso em: 25 maio 2010.

_____. **Quartz core framework reference**. [S.l.], 2010e. Disponível em: <http://developer.apple.com/iphone/library/documentation/GraphicsImaging/Reference/QuartzCoreRefCollection/QuartzCoreRefCollection.pdf>. Acesso em: 25 maio 2010.

_____. **UIKit framework reference**. [S.l.], 2010f. Disponível em: <http://developer.apple.com/iphone/library/documentation/uikit/reference/uikit_framework/UIKit_Framework.pdf>. Acesso em: 25 maio 2010.

_____. **Sample GLImageprocessing**. [S.l.], 2010g. Disponível em: <https://developer.apple.com/iphone/library/samplecode/GLImageProcessing/Introduction/Intro.html>. Acesso em: 27 maio 2010.

_____. **Sample viewTransitions**. [S.l.], 2010h. Disponível em: <https://developer.apple.com/iphone/library/samplecode/ViewTransitions/Introduction/Intro.html >. Acesso em: 27 maio 2010.

_____. **Sample GLPaint**. [S.l.], 2010i. Disponível em: <http://developer.apple.com/iphone/library/samplecode/GLPaint/Introduction/Intro.html >. Acesso em: 27 maio 2010.

CONCI, Aura; AZEVEDO, E. **Computação gráfica**: teoria e prática. 3. ed. Rio de Janeiro: Campus, 2003.

CORREA, Bruno. et al. Análise de imagens médicas via dispositivos móveis. In: WORKSHOP DE INFORMÁTICA MÉDICA, 7., 2007, Porto de Galinhas. **Anais…** Porto de Galinhas: SBC, 2007. p. 231-234.

PAULA FILHO, W. P. **Multimídia**: conceitos e prática. 1, ed. Rio de Janeiro: LTC, 2000.

GOMES, Jonas; VELHO, Luiz. **Fundamentos da computação gráfica**. Rio de Janeiro: IMPA, 2003.

GUIMARÃES, Renato R. **Conversão de imagens do formato DICOM visando a inter-operacionalidade de sistemas através da WEB**. 2002. 113 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

MARK, Dave; LAMARCHE, Jeff. **Beginning iPhone development**: exploring the iPhone SDK. Nova York: Apress, 2009.

MELO Luciana. **Médico x paciente**: tensão nos consultórios. São Paulo, 2009. Disponível em: <http://www.advsaude.com.br/noticias.php?local=1&nid=2175>. Acesso em: 10 mar. 2010.

NEMA. **DICOM - Digital Imaging and COmmunications in Medicine**. Virginia, 2009. Disponível em: <http://medical.nema.org>. Acesso em: 21 mar. 2010.

OSIRIX. **DICOM viewer**. Santa Monica, 2009. Disponível em: <http://www.osirix-viewer.com/>. Acesso em: 21 mar. 2010.

SABBATINI, Renato M. E. **Telemedicina**: a assistência à distância. **Revista Repórter Médico**, São Paulo, v. 2, n. 3, p. 20-22, jun., 1999.

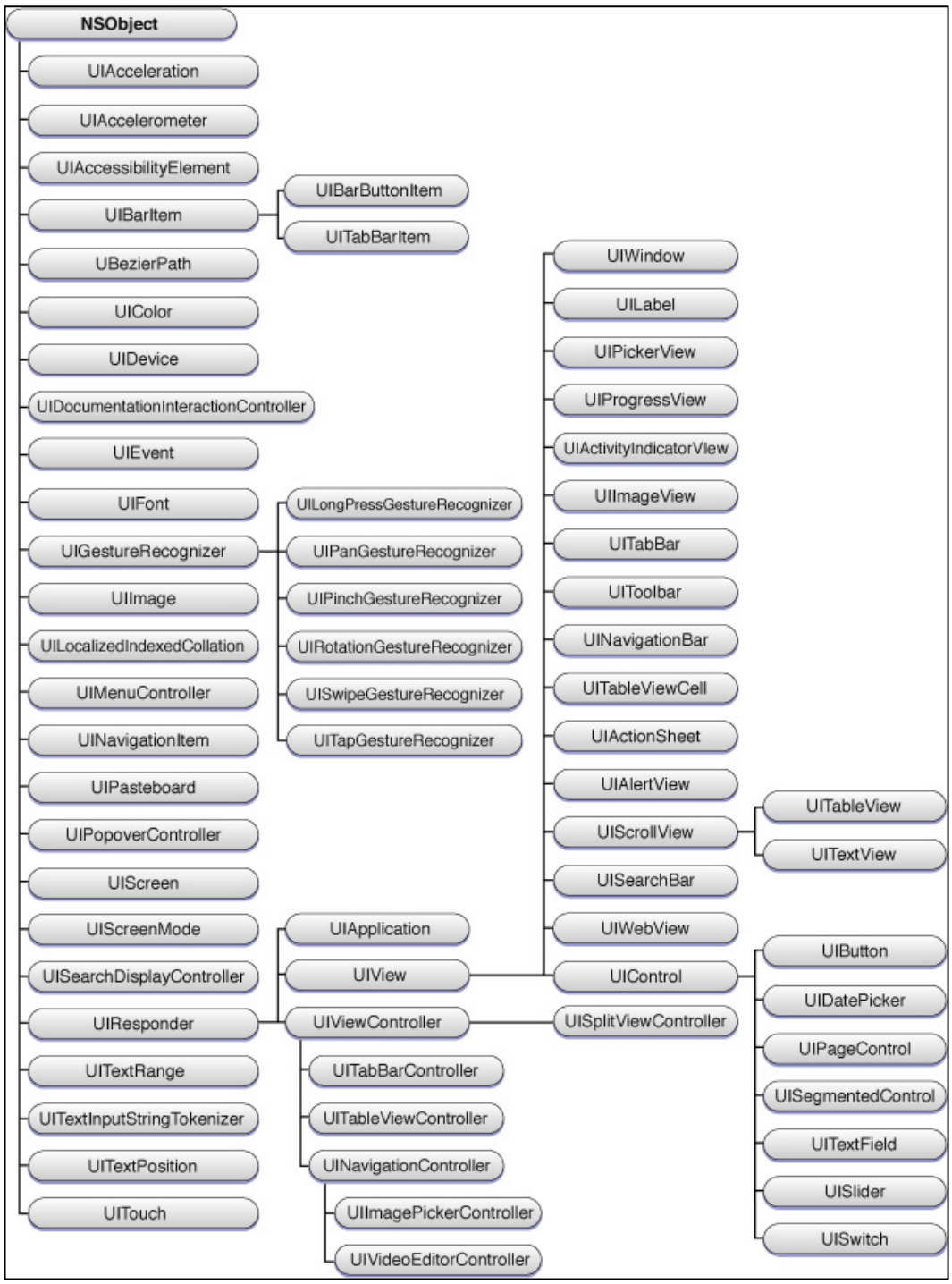
SILVA, Ronaldo G. **Padrão de comunicação de imagens médicas DICOM V3.0**. Salvador, 2002. Disponível em: <http://www.scribd.com/doc/9692832/Padrao-DICOM>. Acesso em: 21 mar. 2010.

SIQUEIRA, Ethevaldo. A mágica... e o mágico. **Revista Veja**, São Paulo, v. 40, n. 2, p. 54-59, mar., 2007.

STIMAC, Gary K. **Introdução ao diagnóstico por imagens**. Rio de Janeiro: Guanabara Koogan, 1992.

ANEXO A – Hierarquia de classes da framework UIKit

No Quadro 13 é mostrada a hierarquia das classes da biblioteca UIKit.



Fonte: Apple (2010f).

Quadro 13 - Hierarquia de classes da framekork UIKit