Sudan University of science &Technology

College Of Graduate Studies

Design of Picture Archiving and Communications System

تصميم نظام أرشفه الصور و الاتصالات

A Thesis Submitted in partial Fulfillment of the Requirements for the M.SC degree in Biomedical Engineering

By:

Izzeldin Amin Mohammed Ali

Supervised by:

Dr. Eltahir Mohamed Hussein

March 2016

ABSTRACT

A DICOM viewer is designed using C# for display digital image , and creates the patient table in the viewer and connected with databases table in the server (SQL server 2008) to resaved and saved the digital image from other modality by Connected the Network [TCP/IP (Transmission Control Protocol/Internet Protocol) Ethernet protocol] , image transfers can be initiated either by the radiological imaging modality (a “push” operation) or by the destination PACS acquisition gateway computer.

المستخلص

صمم برنامج DICOM VIEWER بواسطه لغه ال C# لعرض الصور الرقميه وانشاء جداول للمرضي وربطها بي قواعد البيانات في الخادم لاستقبال و تخزين الصور الرقميه بتوصيل الشبكه (TCP/IP) , الصور المرسله تم انشائها بواسطه بواسطه اجهزه التصوير الطبي او بواسطه اجهزه الPACS الطرفيه .

Chapter one

Introduction

1.1 General view

1.1.1 Picture Archiving and Communications System (PACS)

PACS is used to help hospitals to capture, manage, store, and view diagnostic images. The diagnostic images are generated from modalities and are sent to PACS in electronic form after the scan. Upon receiving the diagnostic images via PACS, technologists may review and validate the images with the performed procedures. After the review and validation, radiologists may start reading the images and performing the diagnosis. This scanning-reviewing-reporting process used to take more than an hour with traditional films. PACS shortens this process to less than five minutes, thereby enabling patients to promptly receive proper treatments. Another benefit of using PACS is time saving, since radiologists no longer need to sort and handle films. Instead, they can now fully devote their times to read images and report the diagnosis using PACS. [1]

1.1.2Modalities

Modality is another name for diagnostic imaging device, which allows physicians to make accurate diagnosis by revealing interiors of the human body without anatomical

Procedures. Various types of modalities can easily be found in modern hospitals, such as MRI, CT, CR, US, PET, and NM. The diagnostic images generated from these modalities have distinctive characteristics. For example, CR focuses the x-rays on the patient plane while CT aims the x-rays on the cross-section plane of the patient. Furthermore, MRI images provide the best contrast between normal and damaged tissues while NM images are ideal for showing the presence and size of abnormalities in the body organ .With the availability of these medical imaging technologies, a physician can request different kinds of scans to obtain anatomic views of the patient in order to make the most accurate diagnosis. Fifteen years ago, radiologists would only receive x-ray films a few hours after the scan was performed. With the introduction of PACS, radiologists and physicians can immediately access these images at a workstation.

1.1.3Digital image and communication in medicine (DICOM)

DICOM stands for Digital Imaging and communications in Medicine and represents years of effort to create the most universal and fundamental standard in digital medical imaging. As such, it provides all the necessary tools for the diagnostically accurate representation and processing of medical imaging data. Moreover, contrary to popular belief, DICOM is not just an image or file format. It is an all-encompassing data transfer, storage, and display protocol built and designed to cover all functional aspects of digital medical imaging (which is why many view DICOM as a set of standards, rather than a single standard). Without a doubt, DICOM truly governs practical digital medicine. Another important acronym that seemingly all DICOM companies plug into their names is PACS (Picture Archiving and Communication Systems). PACS are medical systems (consisting of necessary hardware and software) designed and used to run digital medical imaging. They comprise digital image acquisition devices (modalities – such as computed tomography (CT) scanners, or ultrasound), digital image archives (where the acquired

images are stored), and workstations (where radiologists view the images). When you play with your digital camera (modality), store the images on your computer (archive), and send them to your friends (reviewers), you use the exact same model. Of course, PACS take the model to a much more complex level. [2]

PACS are directly related to DICOM. Their functionality is DICOM-driven, which ensures their interoperability. For that reason, any PACS device or software comes with its own DICOM Conformance Statement, which is a very important document explaining the extent to which the device supports the DICOM standard. In essence, PACS bring the DICOM standard to life.[1]

1.2 problem statement

- The major benefit of PACS resides in its ability to communicate images and reports to referring physicians in a timely and reliable fashion.
- Filmless radiology offers the opportunity to redesign departmental and enterprise-wide workflow with increase.
- Efficiencies of technologists.
- PACS may improve patient care by providing real-time radiology and by enabling teleradiology.
- These technologies and the changing environment in radiology may force the radiologists to become more directly involved in the triage and management decisions for the patient.
- Occasional consultations, regularly scheduled consultation services, and multidisciplinary clinical discussions
- Help directing physicians to the best sequence of examinations to resolve a clinical problem. [3]

1.3 project objectives

The objectives of this research art to

1. Design a model of the PACS system using C# program.
2. Improve the examination and diagnoses.
3. Reduce the film, chemical and cost.

1.4 Thesis layout

This thesis consist of six chapters, chapter on is an introduction, theoretical background in chapter two, the Literature review in chapter three , chapter four describe the proposed system (methodology) , while chapter five represent results and discussion finally the conclusion and recommendation represent in chapter six.

1.5 methodologies

Using C# implanted in the Visual studio 2010 and image process techniques to design PACS workstation to display and adjust the contrast of the DICOM image to view a clearer picture.

Save the image and patient data in the patient table using SQL2008 (PACS Server and Archive) and use the Communications and Networking (TCP/IP communication protocols) to resend the image to destination PACS and exchange the data between

the other systems (Communication by way of media involves the transmission of data and information from one place to another). As figure 1

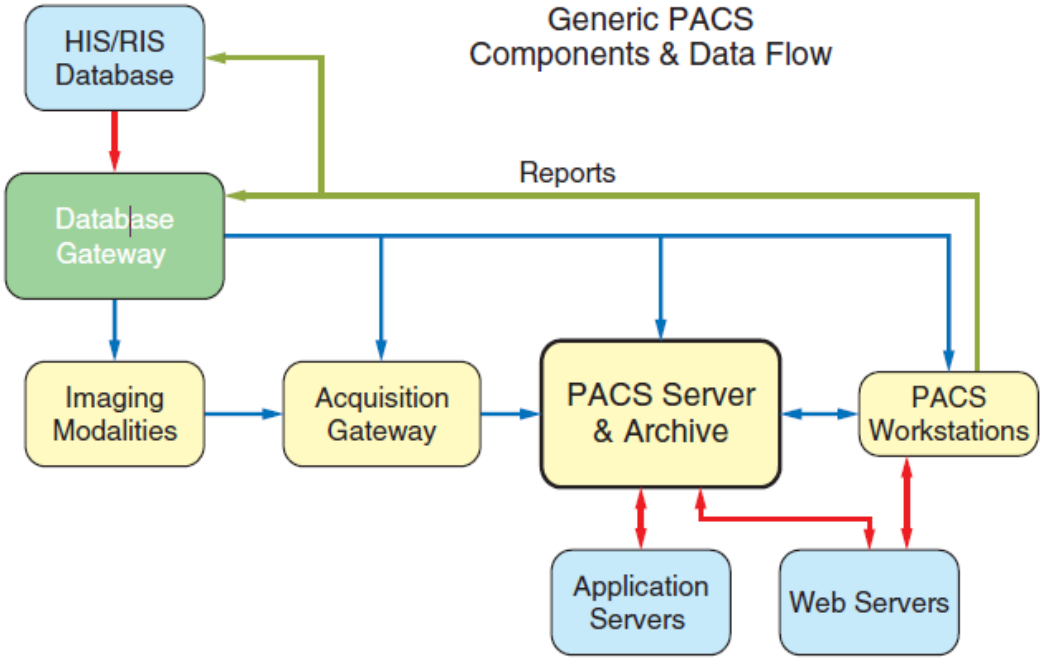

Figure 1 PACS basic components (yellow) and data flow (blue: internal; green and red: external between PACS and other information systems); other information systems (light blue). Application servers and Web servers connected to the PACS controller enrich the PACS infrastructure for other clinical, research, and education applications. HIS: hospital information system; RIS: radiology information system.

Chapter Two
Theoretical Background

2.1 PACS COMPONENTS

The major components in PACS consist of an image and data acquisition gateway, a PACS server and archive, and several display workstations (WSs) integrated together by digital networks. PACS can be further connected with other healthcare information Systems by database gateways and communication networks as shown in Figure 2

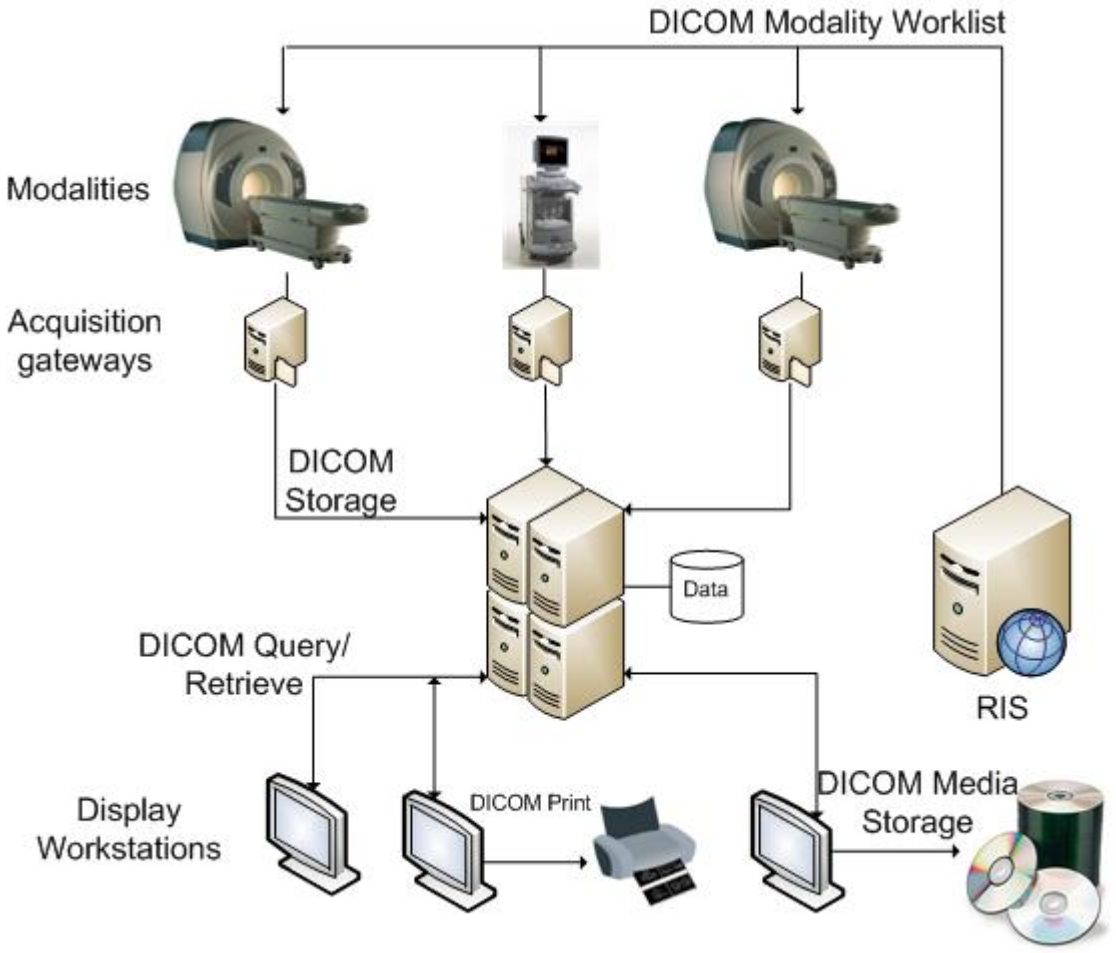

Figure 2 PACS COMPONENTS

2.2 Major Functions of a PACS Controller and Archive Server

• Receives images from examinations (exams) via acquisition gateway computers.
• Extracts text information describing the received exam.
• Updates a network-accessible database management system.
• Determines the destination workstations to which newly generated exams are to be forwarded.
• Automatically retrieves necessary comparison images from a distributed cache storage or long-term library archive system.
• Automatically corrects the orientation of computed radiography images.
• Determines optimal contrast and brightness parameters for image display.
• Performs image data compression if necessary.
• Performs data integrity check if necessary.
• Archives new exams onto long-term archive library.
• Deletes images that have been archived from acquisition gateway computers.
• Services query/retrieve requests from workstations and other PACS controllers in the enterprise PACS.
• Interfaces with PACS application servers.

2.3 Type of Workstation

1. Desktop workstation
   Related information from a Desktop application connected to the PACS server and archive server.
2. Web client to access images

Related information from a Web server connected to the PACS server and archive server.

2.4 Digital Medical Image

A digital image is a two-dimensional array of nonnegative integer function, $f(x,y)$, where $1 \leq x \leq M$ and $1 \leq y \leq N$, and M and N are positive integers representing the number of columns and rows, respectively. For any given x and y, the small square in the image represented by the coordinates (x, y) is called a picture element, or a pixel, and $f(x, y)$ is its corresponding pixel value. If $M = N$, then f becomes a square image; most sectional images in a three-dimensional (3- D) image volume used in medicine are square images. If the digital image $f(x, y, z)$ is 3-D, then the picture element is called a voxel. As $f(x, y, z)$ is collected through time t, the collection becomes a four-dimensional image set where the fourth dimension is t. Throughout this book, we use the symbols f and p interchangeably, f is used when mathematics is presented, and p when picture or image is being emphasized. [3]

2.4.1DICOM BMPs

A digital image, as you might already know, is a rectangular matrix of pixels (picture elements), tiny dots of different colors that form the actual picture. For example, a typical CT image is 512 pixels wide and 512 pixels high; that is, it contains $512 \times 512 = 262{,}144$ pixels. If you write these pixels line-by-line starting from the top left corner you will have a sequence of 262,144 pixel values, which you can store in a file. Essentially, this file is your raw BMP image (Fig. 3).

Image width and height: definitely, you need to know them. Their product is often referred to as spatial image resolution and equals the total number of pixels in the image.

Samples per pixel: each image pixel can be a blend of several sample values. The most typical case is a color pixel, which comprises three independent color samples: red, green, and blue (known as the RGB color space). While the strength of each sample contributes to the pixel's brightness, their mixture creates the color. For example, mixing equal quantities of red and green produces shades of yellow and equal quantities of red, green, and blue correspond to grayscales. Grayscale images, however, are usually stored with one monochrome sample per pixel, corresponding to the pixel's grayscale luminance. When 2-byte (OW VR) samples are used, this provides for 2**2×8** = 65,536 possible grayscales: the source of DICOM support for deep grayscales in CT, MR, CR (X-Ray) and many other modalities. In any event, the choice of pixel sampling remains constant for all pixels in an image, depending only on the imaging technique.[2]

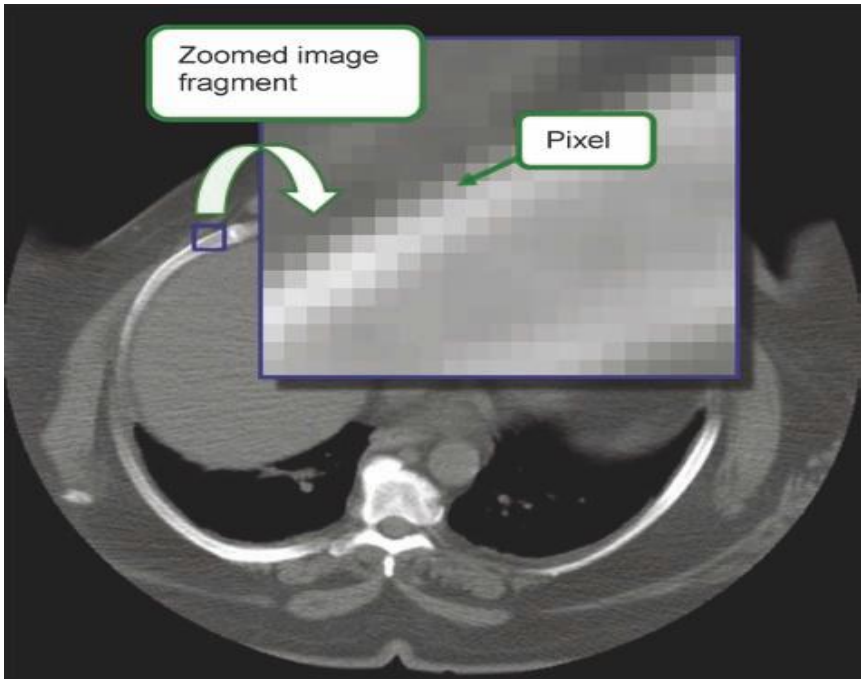

Fig.3 zooming into image pixels

2.4.2 Contrast

Contrast generally refers to the difference in luminance or grey level values in an image and is an important characteristic. It can be defined as the ratio of the maximum intensity to the minimum intensity over an image. Contrast ratio has a strong bearing on the resolving power and detestability of an image. Larger this ratio, more easy it is to interpret the image. Satellite images lack adequate contrast and require contrast improvement.

2.4.2 Contrast enhancement

Contrast enhancement techniques expand the range of brightness values in an image so that the image can be efficiently displayed in a manner desired by the analyst. The density values in a scene are literally pulled farther apart, that is, expanded over a greater range. The effect is to increase the visual contrast between two areas of different uniform densities. This enables the analyst to discriminate easily between areas initially having a small difference in density.[4]

2.5 Visual Studio 2010

Visual Studio 2010 (VS) is an integrated development environment (IDE); a set of tools in a single application that helps you write programs. Without VS, you would need to open a text editor, write all of the code, and then run a command-line compiler to create an executable application. The issue with the text editor and command-line compiler is that you would lose a lot of productivity through manual

processes. Fortunately, you have VS to automate many of the mundane tasks that are required to develop applications.

By knowing the benefits of VS, you have an appreciation for what VS can do for you, increasing your productivity through automatically generated code, rapid coding and visual design, and extensibility .[6]

2.6 C# Programming Languages

The C# programming language was simple and flexible, and it became one of the most popular programming languages in computing. C# is an object-oriented language that allows programmers to organize their software and process the information more effectively than most other programming languages. These programming languages are used extensively in PACS application software packages. [6]

2.7 SQL (Structural Query Language)

SQL is an interface to a relational database such as Oracle. All SQL statements are instructions to operate on the database. Hence SQL is different from general programming languages such as C #. SQL provides a logical way to work with data for all types of users, including programmers, database administrators, and end users. For example, to query a set of rows from a table, the user defines a condition used to search the rows. All rows matching the condition are retrieved in a single step and can be passed as a unit to the user, to another SQL statement, or to

a database application. The user does not need to deal with the rows one by one, or to worry about how the data are physically stored or retrieved. [1]

2.8 Communications Protocol

Images can be missing at the gateway computer when they are transmitted from the imaging device. For example, consider an MRI scanner using the DICOM store client to transfer images with one of the following three modes:

1) The auto-transfer mode transmits an image whenever it is available.
2) The auto queue mode transfers images only when the entire study has been completed.
3) The manual-transfer mode allows the transfer of multiple images, series, or studies.

Under normal operation the auto-transfer mode is routinely used and the manual transfer mode is used only when a retransmission is required. Once the DICOM communication between the scanner and the gateway has been established, if the technologist changes the transfer mode for certain clinical reasons in the middle of the transmission, some images could be temporarily lost.

Chapter Three

Literature review

Digital Imaging and Communication in Medicine (DICOM) protocol has been the standard for processing and administration of medical image data. Picture Archiving and Communication System (PACS) is commonly used in the hospital environment as the tool to manage radiological images which has standardized on the DICOM format. PACS usually consists of dedicated high performance server Computers to provide functionalities of acquisition, storage, retrieval, editing (image and metadata), distribution and presentation. Due to the high cost of the PACS, they are usually only installed in the large hospitals. Clinicians and researchers who work outside the large hospital environment often use a DICOM viewer application to work with images on personal computers or workstations that are not connected to PACS. As the personal computers become more powerful, common desktop workstations can process large amount of image data with performance comparable to the high cost dedicated systems. [8]

All PACS software allowing many functions to be performed at the workstation, because functionality is usually more than adequate, preferences are mostly subjective. It is more important to consider the ease of the software interface with hospital information systems, the radiology information system (RIS), and imaging systems, as well as how the software affects communication between systems. In general, the hospital information system is the source of patient demographic and billing information, the RIS sends imaging requests to the PACS and to the imaging systems, the imaging systems send images to the PACS, and the PACS tells the RIS that the study is complete. Signed reports initially enter the RIS, which forwards the report to the PACS, where the report and images are integrated.

These complex technical details are not discussed here, but they must be carefully designed and specified. [9]

The basic components, standards and corresponding systems of a PACS are very similar. The properties of a PACS, as a system that typically acquire, store, transmit, display, and process digital images. On the basis of this definition the components can be separated. For acquiring and preprocessing of the images an image acquisition component is needed. To store the image a database component exists, controlled by the PACS controller, which is the central controlling component in a PACS. Finally, to display the digital images, a viewing component is required, referred to as workstation.[7]

Chapter Four
The proposed system (Methodology)

4.1 PACS general data flow

(1) RIS notifies the imaging modality and the PACS server of a patient registration. .

(2) Images are sent from the modality to the PACS server.

(3) PACS server archives the images.

(4) Sends to WSs automatically.

(5) Server also prefetches pertinent historical images and sends copies to selected WSs.

(6) images can also be query/retrieve by the WSs .

(7) All WSs have local storage.

(8) Diagnostic reports are sent back to the PACS server or directly to RIS.

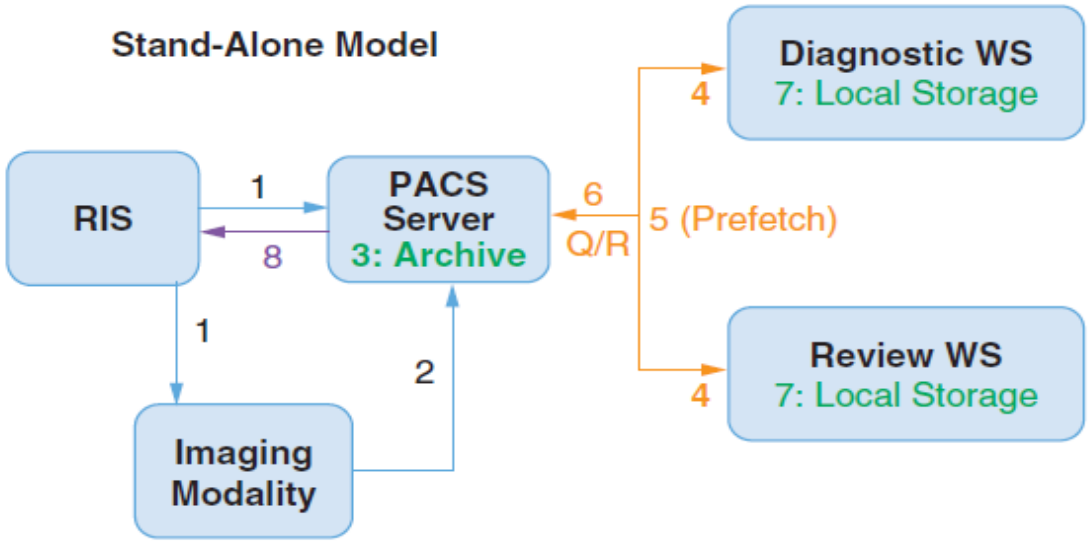

Figure 3 Data workflow of the stand-alone PACS model

4.2 PACS system

The code is written in C#, and built on Visual Studio 2008. The software itself is organized into a set of files AND connects with created databases using SQL 2008. And used Communications and Networking (TCP/IP communication protocols) . Communication by way of media involves the transmission of data and information from one place to another. The media may be bound (cables) or unbound (broadcast, wireless)

The PACS consist of DICOM Viewer, a PACS server and archive and several display workstations (WSs) integrated together by digital networks. PACS can be further connected with other cline work station and connect to PACS server by database gateways to acquisition the image as show on figure 3.

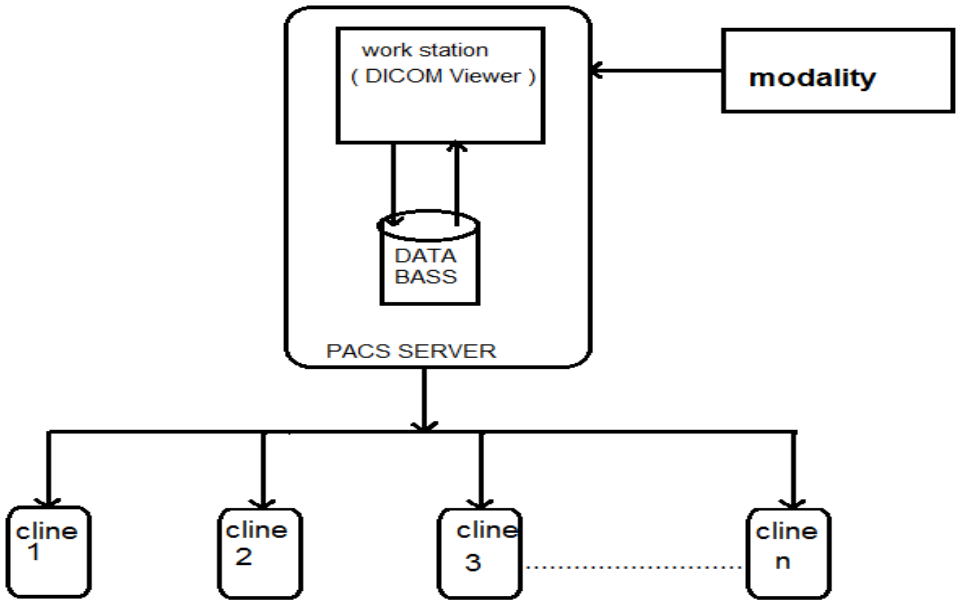

Figure 4 PACS System block diagram

4.2.1 Display Workstations

4.2.1.1 The current DICOM standard includes related to ISO (International Standardization Organization) as:

Part 1: Introduction and Overview

Part 2: Conformance

Part 3: Information Object Définitions

Part 4: Service Class Specifications

Part 5: Data Structures and Encoding

Part 6: Data Dictionary

Part 7: Message Exchange

Part 8: Network Communication Support for Message Exchange

Part 9: Point-to-Point Communication Support for Message Exchange (Retired)

Part 10: Media Storage and File Format for Media Interchange

Part 11: Media Storage Application Profiles

Part 12: Media Formats and Physical Media-for-Media Interchange

Part 13: Print Management Point-to-Point Communication Support (Retired)

Part 14: Gray Scale Standard Display Function

Part 15: Security and System Management Profiles

Part 16: Content Mapping Resource

Part 17: Explanatory Information

Part 18: Web Access to DICOM Persistent Objects (WADO)

4.2.1.2 The software organization into a set of files

The following files are organized as below

1. ***DicomDictionary.cs,*** which contains the DICOM Dictionary Checked with DICOM Standard Part 6.
2. ***DicomDecoder.cs,*** whose main function is to parse the DICOM file and store the necessary attributes appropriately. One of the important methods here is intended to get the next tag:
3. ***ImagePanelControl.cs,*** reused from an earlier article written by us. This image panel contains inbuilt image scrolling should the image size become bigger than the display area.
4. ***WindowLevelGraphControl.cs,*** which has the primary responsibility of displaying the graph control on the screen. An explanation of Window Level.

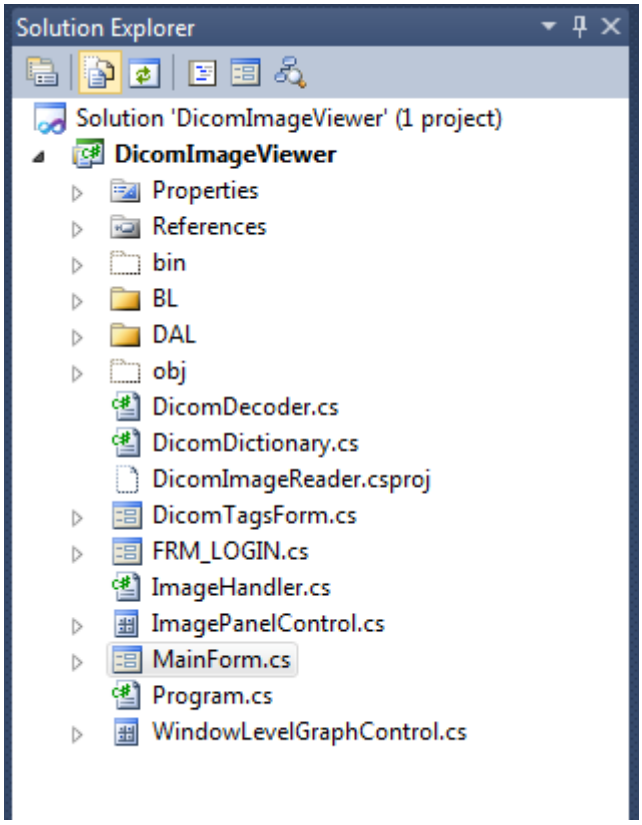

FIGURE 5 DICOM image viewer files

4.2.1.3 DICOM Image File Format

A workstation (WS) includes communication network connection, local database, display, resource management, and processing software. High-quality WSs for radiologists to make primary diagnosis are called diagnostic WS, the others are generally called review WS.

We now present a brief description of the DICOM image file format. As with all other image file formats, a DICOM file consists of a header, followed by pixel data. The header comprises, among other things, the patient name and other patient particulars, and image details. Important among the image details are the image

dimensions - width and height, and image bits per pixel. All of these details are hidden inside the DICOM file in the form of tags and their values.

DICOM objects are standardized according to IODs (**Information Object Definitions**). An IOD is a collection of attributes describing a data object. In other words, an IOD is a data abstraction of a class of similar real-world objects which defines the nature and attributes relevant to that class. DICOM has also standardized on the most commonly used attributes, and these are listed in the DICOM **Data Dictionary** (Part 6 of the Standard). An application which does not find a needed attribute name in this standardized list may add its own private entry, termed as a private tag; proprietary attributes are therefore possible in DICOM.

Characterizing an attribute are its tag, VR, VM (Value Multiplicity), and value. A **tag** is a 4 byte value which uniquely identifies that attribute. A tag is divided into two parts, the Group Tag and the Element Tag, each of which is of length 2 bytes. For example, the tag 0010 0020 (in hexadecimal) represents Patient ID, with a VR of LO (Long String). In this example, 0010 (hex) is the **Group Tag**, and 0020 (hex) is the **Element Tag**. The DICOM Data Dictionary gives a list of all the standardized Group and Element Tags.

One more important concept is **Transfer Syntax**. In simple terms, it tells whether a device can accept the data sent by another device. Each device comes with its own DICOM Conformance Statement, which lists all transfer syntaxes acceptable to the device. A Transfer Syntax tells how the transferred data and messages are encoded. Part 5 of the DICOM Standard gives the Transfer Syntax as a set of encoding rules that allow Application Entities to unambiguously negotiate the encoding techniques (e.g., Data Element structure, byte ordering, compression) they are able to support, thereby allowing these Application Entities to communicate.

(One more term here - **Application Entity** is the name of a DICOM device or program used to uniquely identify it).

The main functionality of a DICOM Image Reader is to read the different tags, as per the Transfer Syntax, and then use these values appropriately. An image viewer needs to read the image attributes - image width, height, bits per pixel, and the actual pixel data. The viewer presented here can be used to view DICOM images with non-compressed transfer syntax. We've made an attempt to read older DICOM files also.

4.2.1.4 DICOM Image Viewer Code

There are a number of freeware DICOM image viewers available. However, we could not find any viewer implemented in C#. Image is a free Java-based viewer (with source code) capable of displaying images of many formats, including DICOM. Our intention here was to emulate the Image code in C#, and create a no-frills simple viewer for DICOM files.

4.2.1.5 The functionality for this viewer is:

- Open DICOM files with Explicit VR and Implicit VR Transfer Syntax
- Read DICOM files where image bit depth is 8 or 16 bits. Also to read RGB DICOM files with bit depth of 8, and 3 bytes per pixel - these images are obtained from the Ultrasound modality.
- Read a DICOM file with just one image inside it
- Read a DICONDE file (a DICONDE file is a DICOM file with NDE - Non Destructive Evaluation - tags inside it)

- Read older DICOM files - some of these do not have the preamble and prefix for. They just contain the string 1.2.840.10008 somewhere in the beginning. For our viewer
- Display the tags in a DICOM file
- Enable the user to view a DICOM image in its entirety - via a "Zoom to Fit" feature
- Enable a user to save a DICOM image as PNG
- Enable to search the patient in the patient table used the ID as primary key.

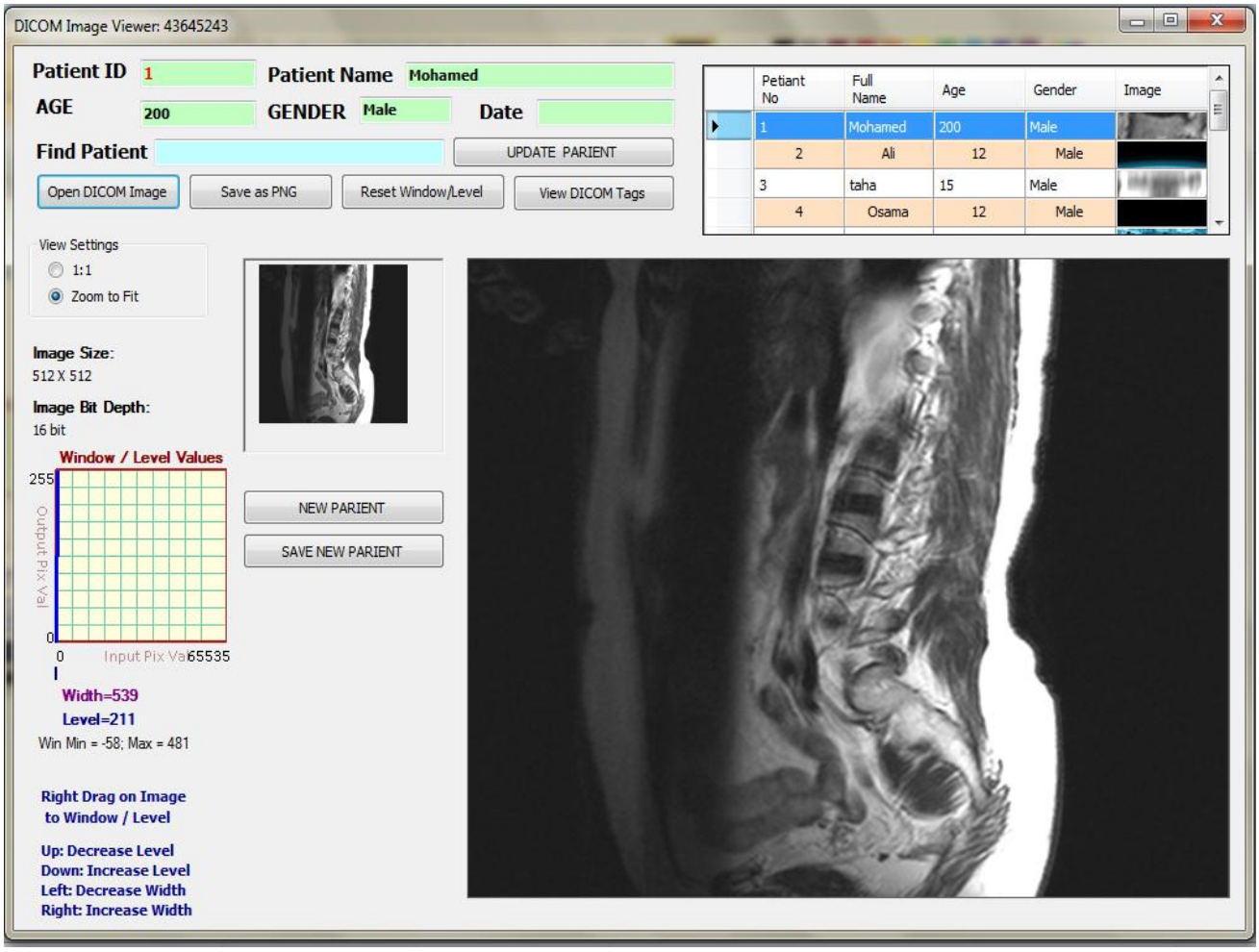

Figure 6 the work station (DICOM viewer) of the PACS

4.2.2 Database server and archiving system

The PACS database ensures that all images are automatically grouped into the correct examination, are chronologically ordered, correctly orientated and labeled, and can be easily retrieved using a variety of criteria (for example, name, hospital number, date, referring clinician, etc). All imaging studies of a patient are immediately available on the PACS which encourages review of examinations with preceding studies and intermodality comparisons".

Therefore a PACS database server should consist of redundant databases with identical reliable commercial database software (e.g Oracle, My SQL), supporting Structured Query Language (SQL). The systems should mirror the data in two database servers, to ensure a stable data handling even in the case of system failure or disk crashes.

The hardware of the database system should use an fast multiple central processing unit and performant interfaces, like SCSI (Small Computer System Interface), S-ATA (Serial-Advanced Technology Attachments) and a fast network interface. With this configuration the system can support parallel processing and a simultaneous transfer of images to different networks or network devices.[7]

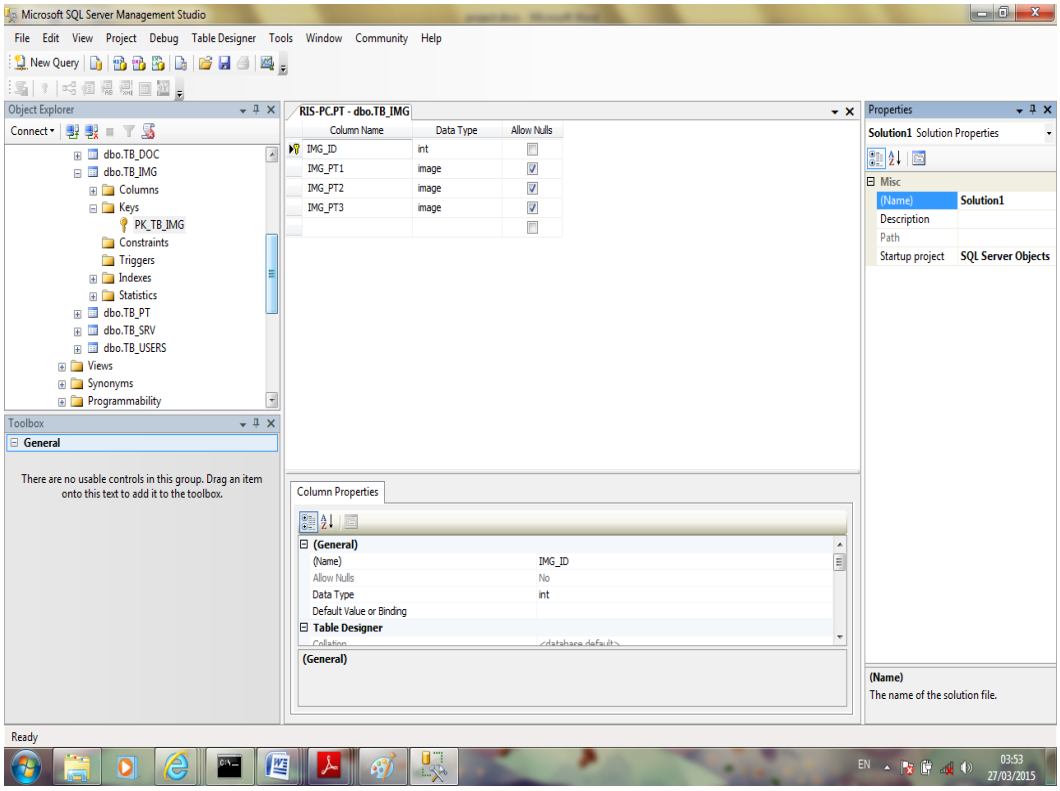

Figure 7 : system databases

4.2.3 PACS Server and Archive

The PACS server and the archive . The former, consisting of the hardware and software architecture, directs the data flow in the entire PACS by using interprocess communication among the major processes. The latter provides a hierarchical image storage management system for short-, medium-, and long-term image archiving. Images arriving at the archive server from various acquisition gateways are not deleted until they have been successfully archived to the permanent storage. On the other hand, all archived images are stacked in the archive server's cache magnetic disks and will be deleted based on their aging criteria (e.g., number of days the examination had been performed; or discharge or transfer of the patient).[1]

4.2.4 Data and Image Acquisition Gateways

The image acquisition gateway computer (gateway) with a set of software programs is used as a buffer between image acquisition and the PACS server. Several acquisition devices (modalities) can share one gateway. [1]

The gateway has three primary tasks

(1) It acquires image data from the radiological imaging device.

(2) It converts the data from manufacturer data format specifications to the PACS image data standard format (header format, byte-ordering, matrix sizes) that is compliant with the DICOM data formats.

(3) It forwards the image study to the PACS server or directly to the PACA workstations (WSs).

4.2.5 DICOM Network

Each device residing on a network is expected to have a network card and its own IP address, which is how the other devices can find it. In addition to that standard networking setup, DICOM assigns to each AE its own DICOM name known as its “Application Entity Title” (AET). In DICOM, the AET is encoded with the AE. The AE can have up to 16 characters. A practical approach for AET naming is to use either the application name (for example, PACSSERVER), or the computer name/location (DRBOBOFFICE) preferably without punctuation signs or

spaces, and in uppercase to avoid ambiguity. This makes it simple to maintain and easy to identify. Unfortunately, certain PACS companies are infamous for using counterintuitive AE titles. When PACS software is installed at your site, make sure the AETs are assigned in a clear and consistent way. [7]

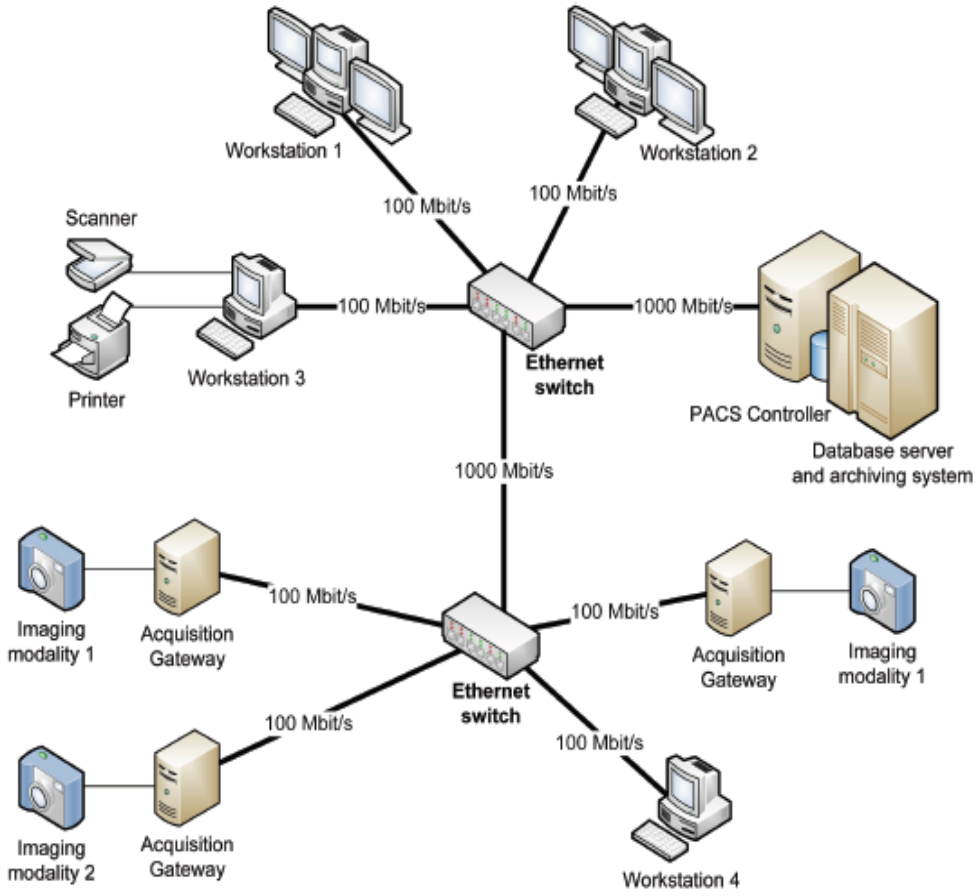

Figure 8: Network model of a PACS

Chapter (5)

Results and discussions

The effect of a Picture Archiving and Communications System (PACS) on diagnostic performance in the accident and emergency department as below

5.1 Results

The results for the comparison of film and PACS periods in terms of patient characteristics are given in tables 2 and 3. The mix of patients presenting at the A&E department was different between the two periods, in two important respects. Firstly, a significantly smaller proportion of patients attending in 1996 were follow up patients and, thus, a significantly larger proportion were presenting with a new problem (table 2). Secondly, a significantly larger proportion of patients attending in 1996 received a radiological examination (table 3). Apart from examinations of an upper limb, for all categories of radiological examinations, a larger proportion of patients received an examination in 1996. This increase was particularly marked for examinations of the chest. In 1992, approximately 6% of all A&E attenders received a chest radiograph, whereas in 1996, approximately 12% of attenders had a chest radiograph.

Table 2

Patient characteristics: comparison in terms of new and follow up A&E attenders

	Film (% of all A&E attenders)	*PACS (% of all A&E attenders)*
Number of new A&E attenders	12 619 (88.52)	15 990 (93.67)
Number of follow up A&E attenders	1 637 (11.48)	1 081 (6.33)
Total number of A&E attenders	14 256	17 071
Proportion of all A&E attenders that were new cases: observed difference between proportions = −0.0515. 95% CI for difference between the proportions is −0.0579 to −0.0451.		

Table 3

Patient characteristics: comparison in terms of body areas examined using radiographic images

Body area	*Film (% all A&E attenders)*	*PACS (% of all A&E attenders)*	*Observed difference between proportions Film-PACS*	*Confidence intervals*
Upper limb	1 106 (7.76)	1 202 (7.04)	−0.00717	−0.0130 to −0.0134
Lower limb	786 (5.51)	1 040 (6.09)	0.00579	0.0006 to 0.0110
Chest	875 (6.14)	2 191 (12.83)	0.0670	0.0606 to 0.0733
Skull	303 (2.13)	509 (2.98)	0.00856	0.00508 to 0.0120
Abdomen	163 (1.14)	635 (3.72)	0.0258	0.0224 to 0.0291
Pelvis	61 (0.43)	128 (0.75)	0.00322	0.00154 to 0.0049
Total radiological exams	3 294	5 705		
Total patients radiographed	2 588 (18.15)	5 345 (31.31)		
Total A&E attenders	14 256	17 071		
Proportion of A&E attenders radiographed: observed difference between the proportions = 0.132. 95% CI for difference between the proportions is 0.122 to 0.141.				

The results for the overall comparison of the film and PACS data collection periods, in terms of misdiagnosis rates, are shown in table 4. During the film period a total of 39 patients were misdiagnosed when film was being used, giving an overall misdiagnosis rate of 1.5% in those patients who were radiographed. The number of patients who were recalled for review (misdiagnosis categories 1 to 3) was 16. During the PACS period a total of 35 patients were misdiagnosed when PACS was being used, giving an overall misdiagnosis rate of 0.66% in those patients who were radiographed. The number of patients in the PACS period who were recalled (misdiagnosis categories 1 to 3) was 20.

Table 4

Misdiagnosis rates: overall comparisons

	Film	PACS
Number of misdiagnoses	39	35
Number of misdiagnoses requiring patient recall	16	20
Number of A&E attenders	14 256	17 071
Number of patients radiographed	2 588	5 345
All misdiagnoses per A&E attender: difference between proportions = −0.000685. 95% CI for difference between the proportions is −0.00178 to 0.000408. All misdiagnoses per radiographed patient: difference between proportions = −0.0085. 95% CI for difference between the proportions is −0.0137 to −0.00335. Misdiagnoses requiring recall per A&E attender: difference between proportions = 0.0000492. 95% CI for difference between the proportions is −0.000703 to 0.000801. Misdiagnoses requiring recall per radiographed patient: difference between proportions = −0.00374. 95% CI for difference between the proportions is −0.00588 to 0.000994.		

The proportion of misdiagnoses among A&E attenders who were radiographed was statistically significantly lower in the period when PACS was being used compared with the period when film was used. However, the proportion of serious misdiagnoses among A&E attenders who were radiographed was not significantly different between the two periods. The data were analysed separately for adults and children (tables 5 and 6). For adults, there was a significantly lower proportion of misdiagnoses overall when PACS was used, but the rate of serious misdiagnoses, requiring patient recall, was the same for the two periods. For children, the misdiagnosis rates, both overall and for serious misdiagnoses, were the same for the PACS and film periods.

Table 5

Misdiagnosis rates: adults (16 years of age and over)

	Film	PACS
Number of misdiagnoses	30	28
Number of misdiagnoses requiring patient recall	12	15
Number of adults radiographed	2155	4474
All misdiagnoses per adult radiographed: difference between proportions = −0.00766. 95% CI for difference between the proportions is −0.0131 to −0.0022. Misdiagnoses requiring recall per adult radiographed: difference between proportions = −0.00222. 95% CI for difference between the proportions is −0.00579 to 0.00135.		

Table 6

Misdiagnosis rates: children (under 16 years of age)

	Film	PACS
Number of misdiagnoses	9	7
Number of misdiagnoses requiring patient recall	4	5
Number of children radiographed	433	871
All misdiagnoses per child radiographed: difference between proportions = −0.0127. 95% CI for difference between the proportions is −0.0274 to 0.00194. Misdiagnoses requiring recall per child radiographed: difference between proportions = −0.0035. 95% CI for difference between the proportions is −0.00138 to 0.00682.		

Tables 7 and 8 and figures 1 and 2 show the distribution and severity of misdiagnoses between body areas, for both adults and children . The results indicate that the misdiagnoses tended to relate, principally, to examinations of the upper or lower limb or skull examinations. There are no pronounced differences in the distributions between the film and PACS periods .

Table 7

Misdiagnoses for adults

	Total radiographed		Number of misdiagnoses (%)	
Body area	**Film**	**PACS**	**Film**	**PACS**
Upper limb	804	845	15 (1.87)	10 (1.18)
Lower limb	687	838	7 (1.02)	12 (1.43)
Chest	794	2045	2 (0.25)	4 (0.20)
Skull	303	376	4 (1.32)	2 (0.53)
Abdomen	163	598	1 (0.61)	0 (0)
Pelvis	61	128	1 (1.64)	0 (0)
Number of adults radiographed	2155	4474	30 (1.39)	28 (0.63)

Table 8

Misdiagnoses in children

	Total radiographed		Number of misdiagnoses (%)	
	Film	**PACS**	**Film**	**PACS**
Upper limb	302	357	6 (1.99)	3 (0.84)
Lower limb	99	202	2 (2.02)	2 (0.99)
Chest	81	146	1 (1.23)	0 (0)
Skull	102	131	0 (0)	2 (1.53)
Number of children radiographed	433	871	9 (2.08)	7 (0.80)

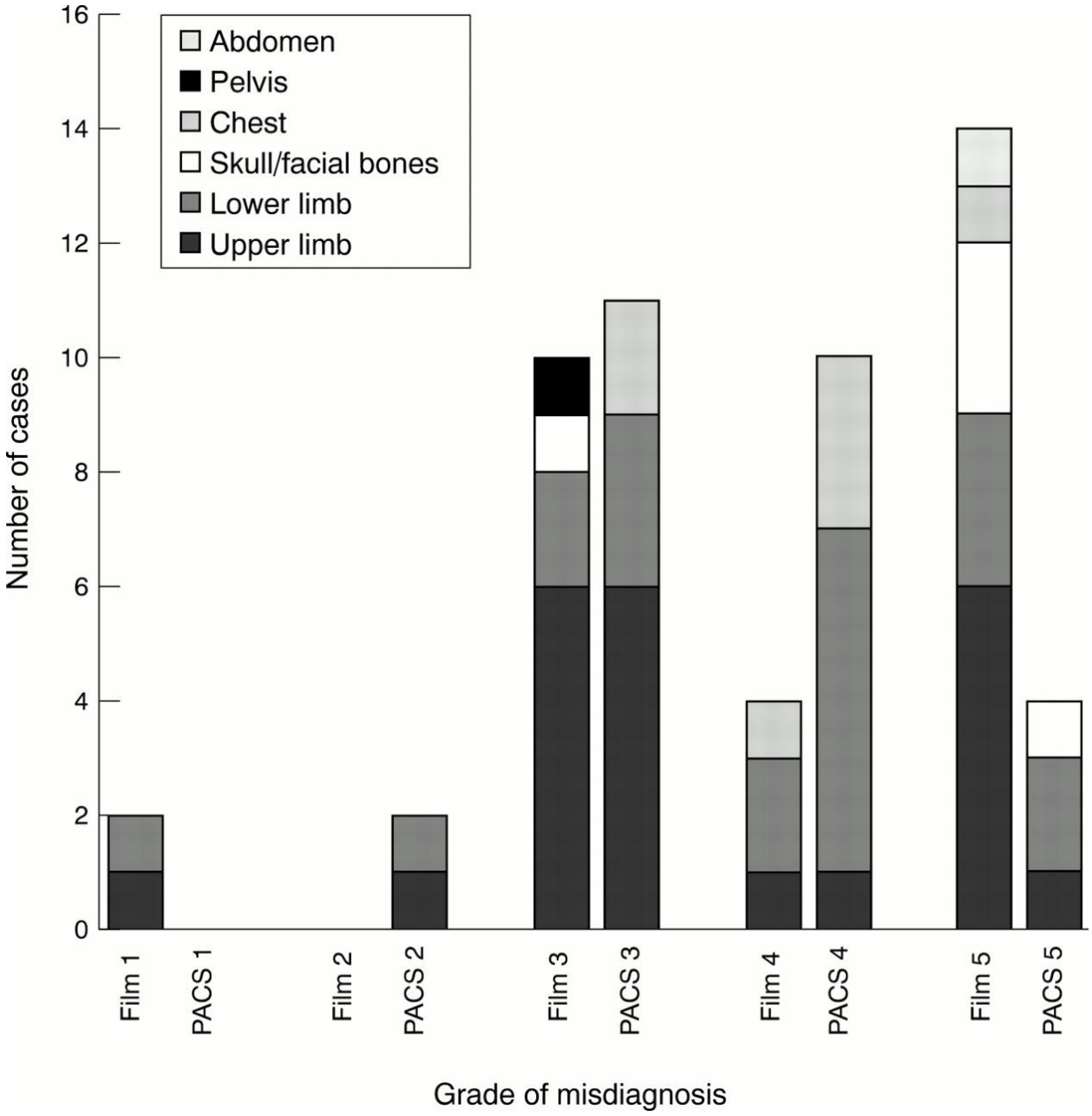

Figure 9

Severity of misdiagnosis by body area for adults .

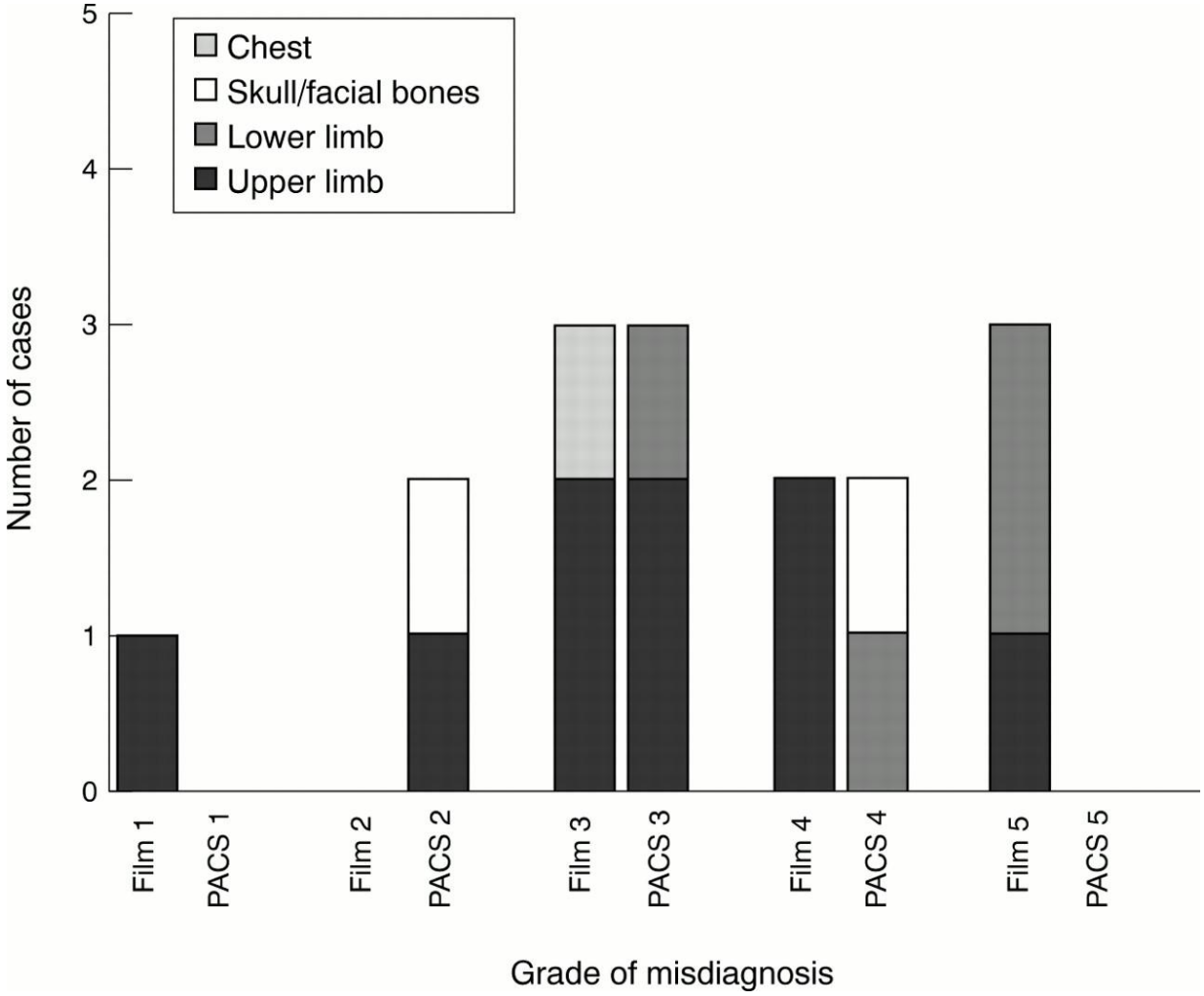

Figure 10

Severity of misdiagnosis by body area for children .

5.2 Discussion

COMPARISON WITH OTHER STUDIES

The number of misdiagnoses by A&E staff identified in this study was very low, for both film and PACS. However, the rates for both film and PACS images identified here are consistent with those reported in other studies.[1-13] In addition these results compare well with a study of conventional radiographic films by Walsh-Kelly *et al* [10] who reported that radiograph misinterpretation by emergency department physicians occurs but is unlikely to result in significant morbidity.

This pragmatic study of two periods of normal mixed workload has shown that A&E clinicians on six month rotations were able to use the new technology without detriment to patient care. It is thus in general agreement with the results of the experimental study by Gillard *et al* [11] who compared the interpretation of digital and film images for skeletal fractures and the acceptance of the technology by A&E staff. In this paper we have not considered the issues of user acceptance and training, but refer readers to other publications about the evaluation of the Hammersmith PACS system, which conclude that the technology has been widely and effectively accepted in the hospital.[12-14]

CONFOUNDING FACTORS THAT OCCURRED OVER THE PERIOD OF THE STUDY

The implementation of the full PACS system occurred over a four year time scale and there were several changes other than the implementation of PACS that may have influenced the results of this study. Inevitably there were changes in personnel in the A&E department but the junior doctors were of the same level of

experience in both periods of the study. The induction training for the SHOs changed between the periods: at the time of the conventional film study (1992), there was a four day induction training course, which included integrated tuition in the use of imaging. By 1996, training was provided on a two day generic A&E course outside the hospital, with specific training in the use of the PACS system held in house. In addition, by the time of the PACS period there were more “middle grade” doctors available to provide advice during normal working hours, but there had been a reduction in consultant presence. Outside normal working hours, including weekends, there was no change in the grade of doctor available.

For the first data collection period, patients were radiographed in the general radiology department on the floor immediately above A&E. In 1993, a dedicated radiology room for A&E patients was built within the A&E department, which was supervised by a superintendent radiographer with specific responsibility for A&E work. The close proximity of a dedicated A&E radiology room is one possible reason for the increase in requests for radiological examinations during the PACS period. The finding of an increase in the use of radiological examinations can be interpreted either as a change in the characteristics of patients seen in the A&E department or as a change in the behavior of A&E clinicians for other reasons, such as an increased fear of litigation. There may be a link between the larger proportion of patients receiving a radiological examination in 1996 and the larger proportion of patients presenting with a new problem.

During the film period, the A&E films were “hot reported” during normal working hours but during the PACS study the images were not routinely reported while the patient was in the A&E department. In 1996 the A&E staff made anecdotal comments that the written radiologists' reports took longer to arrive than in the past. These two factors may have caused the A&E staff to rely less on radiologists and

instead seek the opinions of other senior clinicians or the A&E radiographers who were located nearer than the radiologists. This process of more frequent consultations may have helped to reduce the number of mistakes that were made by inexperienced junior A&E staff. Balancing this effect, PACS provided the potential for images to be viewed simultaneously in A&E and Radiology. It is possible that this may have led to the A&E doctors being more willing to consult radiologists by phone for an opinion, as this would not necessitate a visit to the radiology department. The consequence of this may have been the observed reduction in the number of images misdiagnosed, although we have no data on the number of telephone consultations.

Chapter (6)
Conclusion and future work

6.1 conclusion

PACS are directly derivable, such as the reduction of costs. PACS eliminates the cost for the film roles, big rooms and administrative employees needed for the administration of film-based archives. Moreover, the usage of PACS is increasing the productivity of an imaging department in a hospital or a medical practice, through acceleration of the image workflow. In film-based hospitals the images have to be distributed to the various stations after the acquisition process. In contrast, PACS make it possible to access the studies immediately after acquisition. Consequently the report turnaround time is reduced. However, there also exist plenty of benefits that might be overlooked at the first time: Owing to ordered data and search functions in a PACS, the physicians do not longer have to search for images. As a consequence, the waiting time of patients can be reduced which entails a higher customer satisfaction. Finally, PACS are providing better tools and functionalities at the workstation, improving the job satisfaction of radiologists. As a result fault diagnosis can be reduced. Despite these advantages, there still exist plenty of film-based imaging centers and radiological practices. [7]

6.2 Future work

Digital imaging and computer graphics is still a field with rapid progress, in future aspect

1. Developing higher resolution images.
2. Saved in better data formats with higher compression rates.
3. Besides already realized technologies, such as web access and DICOM web viewers.
4. The connection between the different PACS got to be faced.

References

1. H. K. Huang, D.Sc. FRCR (Hon.), FAIMBE , Copyright (2010)

10. Walsh-Kelly CM, Melzer-Lange MD, Hennes HM, et al. Clinical impact of radiograph misinterpretation in a pediatric ED and the effect of physician training level. Am J Emerg Med1995;13:262–4.

[CrossRef][Medline][Web of Science]

11. Gillard JH, Hubbard C, Das R, *et al*. Digital radiology in skeletal trauma: assessment of casualty officers' performance. *J R Soc Med*1998;**91**:129–2.

[Abstract/FREE Full text]

12. Bryan S, Weatherburn G, Watkins J, *et al. The evaluation of a hospital-wide Picture Archiving and Communications System (PACS). Report to the Department of Health of the Brunel Evaluation of the Hammersmith PACS System.* Uxbridge: Brunel University, Health Economics Research Group, 1998.

13. Watkins J. A hospital wide picture archiving and communication system (PACS) the view of users and providers of the radiology service at Hammersmith Hospital. *Eur J Radiol*1999;**32**:106–12.
[CrossRef][Medline][Web of Science]

14. Weatherburn GC, Watkins J, Bryan S, *et al*. The effect of PACS on the visualization of the lateral cervical spine and the management of patients presenting with trauma. *Med Inf*1997;**22**:359–68.

Appendix (A)

DicomTagsForm.cs

```
using System.Collections.Generic;
using System.Windows.Forms;

namespace DicomImageViewer
{
    public partial class DicomTagsForm : Form
    {
        List<string> str;

        public DicomTagsForm()
        {
            InitializeComponent();
        }

        // Extract the substrings within the DICOM tags main string
        //  to populate the list box
        public void SetString(ref List<string> strg)
        {
            str = strg;
            string s1, s4, s5, s11, s12;

            // Add items to the List View Control
            for (int i = 0; i < str.Count; ++i)
            {
                s1 = str[i];
                ExtractStrings(s1, out s4, out s5, out s11, out s12);

                ListViewItem lvi = new ListViewItem(s11);
                lvi.SubItems.Add(s12);
                lvi.SubItems.Add(s4);
                lvi.SubItems.Add(s5);
                listView.Items.Add(lvi);
            }
        }

        // Saving DICOM tags as Text file
        private void bnSaveAs_Click(object sender, System.EventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "TXT Files(*.txt)|*.txt";
            string s1, s4, s5, s11, s12;

            if (sfd.ShowDialog() == DialogResult.OK)
            {
                System.IO.StreamWriter file = new System.IO.StreamWriter(sfd.FileName);
                for (int i = 0; i < str.Count; ++i)
                {
                    s1 = str[i];
                    ExtractStrings(s1, out s4, out s5, out s11, out s12);
                    file.WriteLine("(" + s11 + "," + s12 + ")\t" + s4 + "\t\t" + s5);
                }
                file.Close();
```

```
            }
        }

        // This method was extracted using the Refactoring facility in Visual Studio
        void ExtractStrings(string s1, out string s4, out string s5, out string s11, out
string s12)
        {
            int ind;
            string s2, s3;
            ind = s1.IndexOf("//");
            s2 = s1.Substring(0, ind);
            s11 = s1.Substring(0, 4);
            s12 = s1.Substring(4, 4);
            s3 = s1.Substring(ind + 2);
            ind = s3.IndexOf(":");
            s4 = s3.Substring(0, ind);
            s5 = s3.Substring(ind + 1);
        }

        private void DicomTagsForm_Load(object sender, System.EventArgs e)
        {

        }
    }
}
```

Appendix(B)

ImageHandler.cs

```
5using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Drawing2D;

namespace ImageProcessing
{
    public class ImageHandler
    {
        private string _bitmapPath;
        private Bitmap _currentBitmap;
        private Bitmap _bitmapbeforeProcessing;
        private Bitmap _bitmapPrevCropArea;

        public ImageHandler()
        {

        }

        public Bitmap CurrentBitmap
        {
            get
            {
                if (_currentBitmap == null)
                    _currentBitmap = new Bitmap(1, 1);
                return _currentBitmap;
            }
            set { _currentBitmap = value; }
        }

        public Bitmap BitmapBeforeProcessing
        {
            get { return _bitmapbeforeProcessing; }
            set { _bitmapbeforeProcessing = value; }
        }

        public string BitmapPath
        {
            get { return _bitmapPath; }
            set { _bitmapPath = value; }
        }

        public enum ColorFilterTypes
        {
            Red,
            Green,
```

```
    Blue
};

public void ResetBitmap()
{
    if (_currentBitmap != null && _bitmapbeforeProcessing != null)
    {
        Bitmap temp = (Bitmap)_currentBitmap.Clone();
        _currentBitmap = (Bitmap)_bitmapbeforeProcessing.Clone();
        _bitmapbeforeProcessing = (Bitmap)temp.Clone();
    }
}

public void SaveBitmap(string saveFilePath)
{
    _bitmapPath = saveFilePath;
    if (System.IO.File.Exists(saveFilePath))
        System.IO.File.Delete(saveFilePath);
    _currentBitmap.Save(saveFilePath);
}

public void ClearImage()
{
    _currentBitmap = new Bitmap(1, 1);
}

public void RestorePrevious()
{
    _bitmapbeforeProcessing = _currentBitmap;
}

public void SetColorFilter(ColorFilterTypes colorFilterType)
{//@@@@@@@@@@@@@@@@@@@@@@@@


    Bitmap temp = (Bitmap)_currentBitmap;
    Bitmap bmap = (Bitmap)temp.Clone();

    //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    Color c;
    for (int i = 0; i < bmap.Width; i++)
    {
        for (int j = 0; j < bmap.Height; j++)
        {
            c = bmap.GetPixel(i, j);
            int nPixelR = 0;
            int nPixelG = 0;
            int nPixelB = 0;
            if (colorFilterType == ColorFilterTypes.Red)
            {
                nPixelR = c.R;
                nPixelG = c.G - 255;
                nPixelB = c.B - 255;
            }
            else if (colorFilterType == ColorFilterTypes.Green)
            {
                nPixelR = c.R - 255;
                nPixelG = c.G;
```

```
                    nPixelB = c.B - 255;
                }
                else if (colorFilterType == ColorFilterTypes.Blue)
                {
                    nPixelR = c.R - 255;
                    nPixelG = c.G - 255;
                    nPixelB = c.B;
                }

                nPixelR = Math.Max(nPixelR, 0);
                nPixelR = Math.Min(255, nPixelR);

                nPixelG = Math.Max(nPixelG, 0);
                nPixelG = Math.Min(255, nPixelG);

                nPixelB = Math.Max(nPixelB, 0);
                nPixelB = Math.Min(255, nPixelB);

                bmap.SetPixel(i, j, Color.FromArgb((byte)nPixelR, (byte)nPixelG,
(byte)nPixelB));
            }
        }
        _currentBitmap = (Bitmap)bmap.Clone();
    }

    public void SetGamma(double red, double green, double blue)
    {
        Bitmap temp = (Bitmap)_currentBitmap;
        Bitmap bmap = (Bitmap)temp.Clone();
        Color c;
        byte[] redGamma = CreateGammaArray(red);
        byte[] greenGamma = CreateGammaArray(green);
        byte[] blueGamma = CreateGammaArray(blue);
        for (int i = 0; i < bmap.Width; i++)
        {
            for (int j = 0; j < bmap.Height; j++)
            {
                c = bmap.GetPixel(i, j);
                bmap.SetPixel(i, j, Color.FromArgb(redGamma[c.R], greenGamma[c.G],
blueGamma[c.B]));
            }
        }
        _currentBitmap = (Bitmap)bmap.Clone();
    }

    private byte[] CreateGammaArray(double color)
    {
        byte[] gammaArray = new byte[256];
        for (int i = 0; i < 256; ++i)
        {
            gammaArray[i] = (byte)Math.Min(255, (int)((255.0 * Math.Pow(i / 255.0,
1.0 / color)) + 0.5));
        }
        return gammaArray;
    }

    public void SetBrightness(int brightness)
    {
```

```
    Bitmap temp = (Bitmap)_currentBitmap;
    Bitmap bmap = (Bitmap)temp.Clone();
    if (brightness < -255) brightness = -255;
    if (brightness > 255) brightness = 255;
    Color c;
    for (int i = 0; i < bmap.Width; i++)
    {
        for (int j = 0; j < bmap.Height; j++)
        {
            c = bmap.GetPixel(i, j);
            int cR = c.R + brightness;
            int cG = c.G + brightness;
            int cB = c.B + brightness;

            if (cR < 0) cR = 1;
            if (cR > 255) cR = 255;

            if (cG < 0) cG = 1;
            if (cG > 255) cG = 255;

            if (cB < 0) cB = 1;
            if (cB > 255) cB = 255;

            bmap.SetPixel(i, j, Color.FromArgb((byte)cR, (byte)cG, (byte)cB));
        }
    }
    _currentBitmap = (Bitmap)bmap.Clone();
}

public void SetContrast(double contrast)
{
    Bitmap temp = (Bitmap)_currentBitmap;
    Bitmap bmap = (Bitmap)temp.Clone();
    if (contrast < -100) contrast = -100;
    if (contrast > 100) contrast = 100;
    contrast = (100.0 + contrast) / 100.0;
    contrast *= contrast;
    Color c;
    for (int i = 0; i < bmap.Width; i++)
    {
        for (int j = 0; j < bmap.Height; j++)
        {
            c = bmap.GetPixel(i, j);
            double pR = c.R / 255.0;
            pR -= 0.5;
            pR *= contrast;
            pR += 0.5;
            pR *= 255;
            if (pR < 0) pR = 0;
            if (pR > 255) pR = 255;

            double pG = c.G / 255.0;
            pG -= 0.5;
            pG *= contrast;
            pG += 0.5;
            pG *= 255;
            if (pG < 0) pG = 0;
            if (pG > 255) pG = 255;
```

```
            double pB = c.B / 255.0;
            pB -= 0.5;
            pB *= contrast;
            pB += 0.5;
            pB *= 255;
            if (pB < 0) pB = 0;
            if (pB > 255) pB = 255;

            bmap.SetPixel(i, j, Color.FromArgb((byte)pR, (byte)pG, (byte)pB));
        }
    }
    _currentBitmap = (Bitmap)bmap.Clone();
}

public void SetGrayscale()
{
    Bitmap temp = (Bitmap)_currentBitmap;
    Bitmap bmap = (Bitmap)temp.Clone();
    Color c;
    for (int i = 0; i < bmap.Width; i++)
    {
        for (int j = 0; j < bmap.Height; j++)
        {
            c = bmap.GetPixel(i, j);
            byte gray = (byte)(.299 * c.R + .587 * c.G + .114 * c.B);

            bmap.SetPixel(i, j, Color.FromArgb(gray, gray, gray));
        }
    }
    _currentBitmap = (Bitmap)bmap.Clone();
}

public void SetInvert()
{
    Bitmap temp = (Bitmap)_currentBitmap;
    Bitmap bmap = (Bitmap)temp.Clone();
    Color c;
    for (int i = 0; i < bmap.Width; i++)
    {
        for (int j = 0; j < bmap.Height; j++)
        {
            c = bmap.GetPixel(i, j);
            bmap.SetPixel(i, j, Color.FromArgb(255 - c.R, 255 - c.G, 255 - c.B));
        }
    }
    _currentBitmap = (Bitmap)bmap.Clone();
}

public void Resize(int newWidth, int newHeight)
{
    if (newWidth != 0 && newHeight != 0)
    {
        Bitmap temp = (Bitmap)_currentBitmap;
        Bitmap bmap = new Bitmap(newWidth, newHeight, temp.PixelFormat);

        double nWidthFactor = (double)temp.Width / (double)newWidth;
        double nHeightFactor = (double)temp.Height / (double)newHeight;
```

```
                double fx, fy, nx, ny;
                int cx, cy, fr_x, fr_y;
                Color color1 = new Color();
                Color color2 = new Color();
                Color color3 = new Color();
                Color color4 = new Color();
                byte nRed, nGreen, nBlue;

                byte bp1, bp2;

                for (int x = 0; x < bmap.Width; ++x)
                {
                    for (int y = 0; y < bmap.Height; ++y)
                    {

                        fr_x = (int)Math.Floor(x * nWidthFactor);
                        fr_y = (int)Math.Floor(y * nHeightFactor);
                        cx = fr_x + 1;
                        if (cx >= temp.Width) cx = fr_x;
                        cy = fr_y + 1;
                        if (cy >= temp.Height) cy = fr_y;
                        fx = x * nWidthFactor - fr_x;
                        fy = y * nHeightFactor - fr_y;
                        nx = 1.0 - fx;
                        ny = 1.0 - fy;

                        color1 = temp.GetPixel(fr_x, fr_y);
                        color2 = temp.GetPixel(cx, fr_y);
                        color3 = temp.GetPixel(fr_x, cy);
                        color4 = temp.GetPixel(cx, cy);

                        // Blue
                        bp1 = (byte)(nx * color1.B + fx * color2.B);

                        bp2 = (byte)(nx * color3.B + fx * color4.B);

                        nBlue = (byte)(ny * (double)(bp1) + fy * (double)(bp2));

                        // Green
                        bp1 = (byte)(nx * color1.G + fx * color2.G);

                        bp2 = (byte)(nx * color3.G + fx * color4.G);

                        nGreen = (byte)(ny * (double)(bp1) + fy * (double)(bp2));

                        // Red
                        bp1 = (byte)(nx * color1.R + fx * color2.R);

                        bp2 = (byte)(nx * color3.R + fx * color4.R);

                        nRed = (byte)(ny * (double)(bp1) + fy * (double)(bp2));

                        bmap.SetPixel(x, y, System.Drawing.Color.FromArgb(255, nRed,
nGreen, nBlue));
                    }
                }
                _currentBitmap = (Bitmap)bmap.Clone();
```

```
            }
        }

        public void RotateFlip(RotateFlipType rotateFlipType)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            bmap.RotateFlip(rotateFlipType);
            _currentBitmap = (Bitmap)bmap.Clone();
        }

        public void Crop(int xPosition, int yPosition, int width, int height)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            if (xPosition + width > _currentBitmap.Width)
                width = _currentBitmap.Width - xPosition;
            if (yPosition + height > _currentBitmap.Height)
                height = _currentBitmap.Height - yPosition;
            Rectangle rect = new Rectangle(xPosition, yPosition, width, height);
            _currentBitmap = (Bitmap)bmap.Clone(rect, bmap.PixelFormat);
        }

        public void DrawOutCropArea(int xPosition, int yPosition, int width, int height)
        {
            _bitmapPrevCropArea = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)_bitmapPrevCropArea.Clone();
            Graphics gr = Graphics.FromImage(bmap);
            Brush cBrush = new Pen(Color.FromArgb(150, Color.White)).Brush;
            Rectangle rect1 = new Rectangle(0, 0, _currentBitmap.Width, yPosition);
            Rectangle rect2 = new Rectangle(0, yPosition, xPosition, height);
            Rectangle rect3 = new Rectangle(0, (yPosition + height),
_currentBitmap.Width, _currentBitmap.Height);
            Rectangle rect4 = new Rectangle((xPosition + width), yPosition,
(_currentBitmap.Width - xPosition - width), height);
            gr.FillRectangle(cBrush, rect1);
            gr.FillRectangle(cBrush, rect2);
            gr.FillRectangle(cBrush, rect3);
            gr.FillRectangle(cBrush, rect4);
            _currentBitmap = (Bitmap)bmap.Clone();
        }

        public void RemoveCropAreaDraw()
        {
            _currentBitmap = (Bitmap)_bitmapPrevCropArea.Clone();
        }

        public void InsertText(string text, int xPosition, int yPosition, string
fontName, float fontSize, string fontStyle, string colorName1, string colorName2)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Graphics gr = Graphics.FromImage(bmap);
            if (string.IsNullOrEmpty(fontName))
                fontName = "Times New Roman";
            if (fontSize.Equals(null))
                fontSize = 10.0F;
            Font font = new Font(fontName, fontSize);
```

```
            if (!string.IsNullOrEmpty(fontStyle))
            {
                FontStyle fStyle = FontStyle.Regular;
                switch (fontStyle.ToLower())
                {
                    case "bold":
                        fStyle = FontStyle.Bold;
                        break;
                    case "italic":
                        fStyle = FontStyle.Italic;
                        break;
                    case "underline":
                        fStyle = FontStyle.Underline;
                        break;
                    case "strikeout":
                        fStyle = FontStyle.Strikeout;
                        break;

                }
                font = new Font(fontName, fontSize, fStyle);
            }
            if (string.IsNullOrEmpty(colorName1))
                colorName1 = "Black";
            if (string.IsNullOrEmpty(colorName2))
                colorName2 = colorName1;
            Color color1 = Color.FromName(colorName1);
            Color color2 = Color.FromName(colorName2);
            int gW = (int)(text.Length * fontSize);
            gW = gW == 0 ? 10 : gW;
            LinearGradientBrush LGBrush = new LinearGradientBrush(new Rectangle(0, 0, gW,
(int)fontSize), color1, color2, LinearGradientMode.Vertical);
            gr.DrawString(text, font, LGBrush, xPosition, yPosition);
            _currentBitmap = (Bitmap)bmap.Clone();
        }

        public void InsertImage(string imagePath, int xPosition, int yPosition)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Graphics gr = Graphics.FromImage(bmap);
            if (!string.IsNullOrEmpty(imagePath))
            {
                Bitmap i_bitmap = (Bitmap)Bitmap.FromFile(imagePath);
                Rectangle rect = new Rectangle(xPosition, yPosition, i_bitmap.Width,
i_bitmap.Height);
                gr.DrawImage(Bitmap.FromFile(imagePath), rect);
            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }

        public void InsertShape(string shapeType, int xPosition, int yPosition, int
width, int height, string colorName)
        {
            Bitmap temp = (Bitmap)_currentBitmap;
            Bitmap bmap = (Bitmap)temp.Clone();
            Graphics gr = Graphics.FromImage(bmap);
            if (string.IsNullOrEmpty(colorName))
                colorName = "Black";
```

```
            Pen pen = new Pen(Color.FromName(colorName));
            switch (shapeType.ToLower())
            {
                case "filledellipse":
                    gr.FillEllipse(pen.Brush, xPosition, yPosition, width, height);
                    break;
                case "filledrectangle":
                    gr.FillRectangle(pen.Brush, xPosition, yPosition, width, height);
                    break;
                case "ellipse":
                    gr.DrawEllipse(pen, xPosition, yPosition, width, height);
                    break;
                case "rectangle":
                default:
                    gr.DrawRectangle(pen, xPosition, yPosition, width, height);
                    break;

            }
            _currentBitmap = (Bitmap)bmap.Clone();
        }
    }
}
```

Appendix (E)

ImagePanelControl.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Imaging;

namespace DicomImageViewer
{
    public partial class ImagePanelControl : UserControl
    {
        List<byte> pix8;
        List<ushort> pix16;
        List<byte> pix24;
        Bitmap bmp;
        int hOffset;
        int vOffset;
        int hMax;
        int vMax;
        int imgWidth;
        int imgHeight;
        int panWidth;
```

```
int panHeight;
bool newImage;

// For Window Level
int winMin;
int winMax;
int winCentre;
int winWidth;
int winWidthBy2;
int deltaX;
int deltaY;

Point ptWLDown;
double changeValWidth;
double changeValCentre;
bool rightMouseDown;
bool imageAvailable;
bool signed16Image;

byte[] lut8;
byte[] lut16;
byte[] imagePixels8;
byte[] imagePixels16;
byte[] imagePixels24;
int sizeImg;
int sizeImg3;
MainForm mf;

ImageBitsPerPixel bpp;

public ViewSettings viewSettings;
public bool viewSettingsChanged;

public ImagePanelControl()
{
    InitializeComponent();
    DoubleBuffered = true;

    pix8 = new List<byte>();
    pix16 = new List<ushort>();
    pix24 = new List<byte>();

    this.hScrollBar.Visible = false;
    this.vScrollBar.Visible = false;

    winMin = 0;
    winMax = 65535;

    ptWLDown = new Point();
    changeValWidth = 0.5;
    changeValCentre = 20.0;
    rightMouseDown = false;
    imageAvailable = false;
    signed16Image = false;

    lut8 = new byte[256];
    lut16 = new byte[65536];
```

```
            viewSettings = ViewSettings.ZoomToFit;
            viewSettingsChanged = false;

            PerformResize();
        }

        public bool NewImage
        {
            set
            {
                newImage = value;
            }
        }

        public bool Signed16Image
        {
            set { signed16Image = value; }
        }

        public void SetParameters(ref List<byte> arr, int wid, int hei, double
windowWidth,
            double windowCentre, int samplesPerPixel, bool resetScroll, MainForm mainFrm)
        {
            if (samplesPerPixel == 1)
            {
                bpp = ImageBitsPerPixel.Eight;
                imgWidth = wid;
                imgHeight = hei;
                winWidth = Convert.ToInt32(windowWidth);
                winCentre = Convert.ToInt32(windowCentre);
                changeValWidth = 0.1;
                changeValCentre = 20.0;
                sizeImg = imgWidth * imgHeight;
                sizeImg3 = sizeImg * 3;

                pix8 = arr;
                imagePixels8 = new byte[sizeImg3];

                mf = mainFrm;
                imageAvailable = true;
                if (bmp != null)
                    bmp.Dispose();
                ResetValues();
                ComputeLookUpTable8();
                bmp = new Bitmap(imgWidth, imgHeight,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                CreateImage8();
            }

            if (samplesPerPixel == 3)
            {
                bpp = ImageBitsPerPixel.TwentyFour;
                imgWidth = wid;
                imgHeight = hei;
                winWidth = Convert.ToInt32(windowWidth);
                winCentre = Convert.ToInt32(windowCentre);
                changeValWidth = 0.1;
                changeValCentre = 0.1;
```

```
                sizeImg = imgWidth * imgHeight;
                sizeImg3 = sizeImg * 3;

                pix24 = arr;
                imagePixels24 = new byte[sizeImg3];

                mf = mainFrm;
                imageAvailable = true;
                if (bmp != null)
                    bmp.Dispose();
                ResetValues();
                ComputeLookUpTable8();
                bmp = new Bitmap(imgWidth, imgHeight,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                CreateImage24();

            }
            if (resetScroll == true) ComputeScrollBarParameters();
            Invalidate();
        }

        void DetermineMouseSensitivity()
        {
            // Modify the 'sensitivity' of the mouse based on the current window width
            if (winWidth < 10)
            {
                changeValWidth = 0.1;
            }
            else if (winWidth >= 20000)
            {
                changeValWidth = 40;
            }
            else
            {
                changeValWidth = 0.1 + (winWidth - 10) / 500.0;
            }

            changeValCentre = changeValWidth;
        }

        public void SetParameters(ref List<ushort> arr, int wid, int hei, double
windowWidth,
            double windowCentre, bool resetScroll, MainForm mainFrm)
        {
            bpp = ImageBitsPerPixel.Sixteen;
            imgWidth = wid;
            imgHeight = hei;
            winWidth = Convert.ToInt32(windowWidth);
            winCentre = Convert.ToInt32(windowCentre);

            sizeImg = imgWidth * imgHeight;
            sizeImg3 = sizeImg * 3;
            double sizeImg3By4 = sizeImg3 / 4.0;

            DetermineMouseSensitivity();

            pix16 = arr;
            imagePixels16 = new byte[sizeImg3];
```

```
            mf = mainFrm;
            imageAvailable = true;
            if (bmp != null)
                bmp.Dispose();
            ResetValues();
            ComputeLookUpTable16();
            bmp = new Bitmap(imgWidth, imgHeight,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            CreateImage16();
            if (resetScroll == true) ComputeScrollBarParameters();
            Invalidate();

        }

        // Create a bitmap on the fly, using 8-bit grayscale pixel data
        private void CreateImage8()
        {
            BitmapData bmd = bmp.LockBits(new Rectangle(0, 0, imgWidth, imgHeight),
                System.Drawing.Imaging.ImageLockMode.ReadOnly, bmp.PixelFormat);

            unsafe
            {
                int pixelSize = 3;
                int i, j, j1, i1;
                byte b;

                for (i = 0; i < bmd.Height; ++i)
                {
                    byte* row = (byte*)bmd.Scan0 + (i * bmd.Stride);
                    i1 = i * bmd.Width;

                    for (j = 0; j < bmd.Width; ++j)
                    {
                        b = lut8[pix8[i * bmd.Width + j]];
                        j1 = j * pixelSize;
                        row[j1] = b;            // Red
                        row[j1 + 1] = b;        // Green
                        row[j1 + 2] = b;        // Blue
                    }
                }
            }
            bmp.UnlockBits(bmd);
        }

        // Create a bitmap on the fly, using 24-bit RGB pixel data
        private void CreateImage24()
        {
            {
                int numBytes = imgWidth * imgHeight * 3;
                int j;
                int i, i1;

                BitmapData bmd = bmp.LockBits(new Rectangle(0, 0, bmp.Width,
                    bmp.Height), ImageLockMode.WriteOnly, bmp.PixelFormat);

                int width3 = bmd.Width * 3;
```

```
        unsafe
        {
            for (i = 0; i < bmd.Height; ++i)
            {
                byte* row = (byte*)bmd.Scan0 + (i * bmd.Stride);
                i1 = i * bmd.Width * 3;

                for (j = 0; j < width3; j += 3)
                {
                    // Windows uses little-endian, so the RGB data is
                    //  actually stored as BGR
                    row[j + 2] = lut8[pix24[i1 + j]];     // Blue
                    row[j + 1] = lut8[pix24[i1 + j + 1]]; // Green
                    row[j] = lut8[pix24[i1 + j + 2]];     // Red
                }
            }
        }
        bmp.UnlockBits(bmd);
    }
}

// Create a bitmap on the fly, using 16-bit grayscale pixel data
private void CreateImage16()
{
    BitmapData bmd = bmp.LockBits(new Rectangle(0, 0, imgWidth, imgHeight),
       System.Drawing.Imaging.ImageLockMode.ReadOnly, bmp.PixelFormat);

    unsafe
    {
        int pixelSize = 3;
        int i, j, j1, i1;
        byte b;

        for (i = 0; i < bmd.Height; ++i)
        {
            byte* row = (byte*)bmd.Scan0 + (i * bmd.Stride);
            i1 = i * bmd.Width;

            for (j = 0; j < bmd.Width; ++j)
            {
                b = lut16[pix16[i * bmd.Width + j]];
                j1 = j * pixelSize;
                row[j1] = b;            // Red
                row[j1 + 1] = b;        // Green
                row[j1 + 2] = b;        // Blue
            }
        }
    }
    bmp.UnlockBits(bmd);
}

private void ComputeScrollBarParameters()
{
    panWidth = panel.Width;
    panHeight = panel.Height;

    hOffset = (panWidth - imgWidth) / 2;
    vOffset = (panHeight - imgHeight) / 2;
```

```
            if (imgWidth < panWidth)
            {
                hScrollBar.Visible = false;
            }
            else
            {
                hScrollBar.Visible = true;
                hScrollBar.Value = (hScrollBar.Maximum + 1 -
                    hScrollBar.LargeChange - hScrollBar.Minimum) / 2;
            }

            if (imgHeight < panHeight)
            {
                vScrollBar.Visible = false;
            }
            else
            {
                vScrollBar.Visible = true;
                vScrollBar.Value = (vScrollBar.Maximum + 1 -
                    vScrollBar.LargeChange - vScrollBar.Minimum) / 2;
            }
        }

        private void vScrollBar_Scroll(object sender, ScrollEventArgs e)
        {
            int val = vScrollBar.Value;
            vOffset = (panHeight - imgHeight) * (val - vScrollBar.Minimum) /
                    (vMax - vScrollBar.Minimum);
            Invalidate();
        }

        private void hScrollBar_Scroll(object sender, ScrollEventArgs e)
        {
            int val = hScrollBar.Value;
            hOffset = (panWidth - imgWidth) * (val - hScrollBar.Minimum) /
                (hMax - hScrollBar.Minimum);
            Invalidate();
        }

        void SetScrollVisibility()
        {
            if (imgWidth >= panWidth)
            {
                hScrollBar.Visible = true;
            }

            if (imgHeight >= panHeight)
            {
                vScrollBar.Visible = true;
            }
        }

        private void ImagePanel_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = Graphics.FromHwnd(panel.Handle);
            g.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBilinear;
```

```
            if (viewSettingsChanged || newImage)
                g.Clear(SystemColors.Control);

            if (((bpp == ImageBitsPerPixel.Eight) && (pix8.Count > 0)) ||
                 ((bpp == ImageBitsPerPixel.Sixteen) && (pix16.Count > 0)) ||
                 ((bpp == ImageBitsPerPixel.TwentyFour) && (pix24.Count > 0)))
            {
                if (viewSettings == ViewSettings.Zoom1_1)
                {
                    SetScrollVisibility();
                    g.DrawImage(bmp, hOffset, vOffset);
                }
                else // if(viewSettings == ViewSettings.ZoomToFit)
                {
                    ScaleImageKeepingAspectRatio(ref g, bmp, panWidth, panHeight);
                }
            }
            g.Dispose();
            viewSettingsChanged = false;
            newImage = false;
        }


        void ScaleImageKeepingAspectRatio(ref Graphics grfx, Image image, int panelWidth,
int panelHeight)
        {
            hScrollBar.Visible = false;
            vScrollBar.Visible = false;
            SizeF sizef = new SizeF(image.Width / image.HorizontalResolution,
                                    image.Height / image.VerticalResolution);

            float fScale = Math.Min(panelWidth / sizef.Width,
                                    panelHeight / sizef.Height);

            sizef.Width *= fScale;
            sizef.Height *= fScale;

            grfx.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
            grfx.DrawImage(image, (panelWidth - sizef.Width) / 2,
                                  (panelHeight - sizef.Height) / 2,
                                  sizef.Width, sizef.Height);
            mf.picbox1.Image = (Bitmap)image.Clone();//
        }

        // Method to save an image as PNG. The image is saved as per the current
window/level values.
        public void SaveImage(String fileName)
        {
            if (bmp != null)
                bmp.Save(fileName, ImageFormat.Png);
        }

        // We use the linear interpolation method here
        //  Nonlinear methods like sigmoid are also common, but we don't do them here.
        private void ComputeLookUpTable8()
        {
```

```
    if (winMax == 0)
        winMax = 255;

    int range = winMax - winMin;
    if (range < 1) range = 1;
    double factor = 255.0 / range;

    for (int i = 0; i < 256; ++i)
    {
        if (i <= winMin)
            lut8[i] = 0;
        else if (i >= winMax)
            lut8[i] = 255;
        else
        {
            lut8[i] = (byte)((i - winMin) * factor);
        }
    }
}

// Linear interpolation here too
private void ComputeLookUpTable16()
{
    int range = winMax - winMin;
    if (range < 1) range = 1;
    double factor = 255.0 / range;
    int i;

    for (i = 0; i < 65536; ++i)
    {
        if (i <= winMin)
            lut16[i] = 0;
        else if (i >= winMax)
            lut16[i] = 255;
        else
        {
            lut16[i] = (byte)((i - winMin) * factor);
        }
    }
}

private void panel_MouseDown(object sender, MouseEventArgs e)
{
    if (imageAvailable == true)
    {
        if (e.Button == MouseButtons.Right)
        {
            ptWLDown.X = e.X;
            ptWLDown.Y = e.Y;
            rightMouseDown = true;
            Cursor = Cursors.Hand;
        }
    }
}

// Mouse-move is made to perform window-level
private void panel_MouseMove(object sender, MouseEventArgs e)
{
```

```
        DetermineMouseSensitivity();
        if (rightMouseDown == true)
        {
            winWidthBy2 = winWidth / 2;
            winWidth = winMax - winMin;
            winCentre = winMin + winWidthBy2;

            deltaX = (int)((ptWLDown.X - e.X) * changeValWidth);
            deltaY = (int)((ptWLDown.Y - e.Y) * changeValCentre);

            winCentre -= deltaY;
            winWidth -= deltaX;

            if (winWidth < 2) winWidth = 2;
            winWidthBy2 = winWidth / 2;

            winMax = winCentre + winWidthBy2;
            winMin = winCentre - winWidthBy2;

            if (winMin >= winMax) winMin = winMax - 1;
            if (winMax <= winMin) winMax = winMin + 1;

            ptWLDown.X = e.X;
            ptWLDown.Y = e.Y;

            UpdateMainForm();
            if (bpp == ImageBitsPerPixel.Eight)
            {
                ComputeLookUpTable8();
                CreateImage8();
            }
            else if (bpp == ImageBitsPerPixel.Sixteen)
            {
                ComputeLookUpTable16();
                CreateImage16();
            }
            else // (bpp == ImageBitsPerPixel.TwentyFour)
            {
                ComputeLookUpTable8();
                CreateImage24();
            }

            Invalidate();
        }
    }

    private void panel_MouseUp(object sender, MouseEventArgs e)
    {
        if (rightMouseDown == true)
        {
            rightMouseDown = false;
            Cursor = Cursors.Default;
        }
    }

    // Update the graph control on the main form
    private void UpdateMainForm()
    {
```

```
            mf.UpdateWindowLevel(winWidth, winCentre, bpp);
        }

        // Restore original window/level values
        public void ResetValues()
        {
            winMax = Convert.ToInt32(winCentre + 0.5 * winWidth);
            winMin = winMax - winWidth;
            UpdateMainForm();
        }

        private void ImagePanelControl_Resize(object sender, EventArgs e)
        {
            PerformResize();
        }

        private void PerformResize()
        {
            panel.Location = new Point(3, 3);
            panel.Width = ClientRectangle.Width - 24;
            panel.Height = ClientRectangle.Height - 24;

            vScrollBar.Location = new Point(ClientRectangle.Width - 19, 3);
            vScrollBar.Height = panel.Height;

            hScrollBar.Location = new Point(3, ClientRectangle.Height - 19);
            hScrollBar.Width = panel.Width;

            hMax = hScrollBar.Maximum - hScrollBar.LargeChange + hScrollBar.SmallChange;
            vMax = vScrollBar.Maximum - vScrollBar.LargeChange + vScrollBar.SmallChange;
        }

        private void panel_Paint(object sender, PaintEventArgs e)
        {

        }
    }
}
```

Appendix (C)

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Linq;
using System.Drawing;
using System.IO;
using ImageProcessing;
using System.ComponentModel;
using System.Data;
```

```
namespace DicomImageViewer
{
    public enum ImageBitsPerPixel { Eight, Sixteen, TwentyFour };
    public enum ViewSettings { Zoom1_1, ZoomToFit };


    /// </summary>
    public partial class MainForm : Form
    {
        BL.CLS_PT izpt = new BL.CLS_PT(); // this to fill data grid with info from DB
        double zoomFactor = 1.0;

        ImageHandler imageHandler = new ImageHandler();// new test
        //

        DicomDecoder dd;
        List<byte> pixels8;
        List<ushort> pixels16;
        List<byte> pixels24; // 30 July 2010
        int imageWidth;
        int imageHeight;
        int bitDepth;
        int samplesPerPixel;  // Updated 30 July 2010
        bool imageOpened;
        double winCentre;
        double winWidth;
        bool signedImage;
        int maxPixelValue;    // Updated July 2012
        int minPixelValue;

        public MainForm()
        {
            InitializeComponent();
            this.dataGridView1.DataSource = izpt.izzpt();// this to get data from stored
proc in sql
            dd = new DicomDecoder();
            pixels8 = new List<byte>();
            pixels16 = new List<ushort>();
            pixels24 = new List<byte>();
            imageOpened = false;
            signedImage = false;
            maxPixelValue = 0;
            minPixelValue = 65535;


        }

        private void bnOpen_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "All DICOM Files(*.*)|*.*";
            if (ofd.ShowDialog() == DialogResult.OK)
            {
                if (ofd.FileName.Length > 0)
                {
                    //***********************************************
                    imageHandler.CurrentBitmap = (Bitmap)picbox1.Image;
                    this.AutoScroll = true;
```

```
                    this.AutoScrollMinSize = new
Size(Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor));

                    //***********************************************
                    Cursor = Cursors.WaitCursor;
                    ReadAndDisplayDicomFile(ofd.FileName, ofd.SafeFileName);
                    imageOpened = true;
                    Cursor = Cursors.Default;

                }
                ofd.Dispose();

            }
        }

        //************************************
        private void ImageProcessing_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.DrawImage(imageHandler.CurrentBitmap, new
Rectangle(this.AutoScrollPosition.X, this.AutoScrollPosition.Y,
Convert.ToInt32(imageHandler.CurrentBitmap.Width * zoomFactor),
Convert.ToInt32(imageHandler.CurrentBitmap.Height * zoomFactor)));
        }

        //************************************
        private void ReadAndDisplayDicomFile(string fileName, string fileNameOnly)
        {
            dd.DicomFileName = fileName;

            TypeOfDicomFile typeOfDicomFile = dd.typeofDicomFile;

            if (typeOfDicomFile == TypeOfDicomFile.Dicom3File ||
                typeOfDicomFile == TypeOfDicomFile.DicomOldTypeFile)
            {
                imageWidth = dd.width;
                imageHeight = dd.height;
                bitDepth = dd.bitsAllocated;
                winCentre = dd.windowCentre;
                winWidth = dd.windowWidth;
                samplesPerPixel = dd.samplesPerPixel;
                signedImage = dd.signedImage;

                label1.Visible = true;
                label2.Visible = true;
                label3.Visible = true;
                label4.Visible = true;
                bnSave.Enabled = true;
                bnTags.Enabled = true;
                bnResetWL.Enabled = true;
                label2.Text = imageWidth.ToString() + " X " + imageHeight.ToString();
                if (samplesPerPixel == 1)
                    label4.Text = bitDepth.ToString() + " bit";
                else
                    label4.Text = bitDepth.ToString() + " bit, " + samplesPerPixel +
                        " samples per pixel";
```

```
imagePanelControl.NewImage = true;
Text = "DICOM Image Viewer: " + fileNameOnly;

if (samplesPerPixel == 1 && bitDepth == 8)
{
    pixels8.Clear();
    pixels16.Clear();
    pixels24.Clear();
    dd.GetPixels8(ref pixels8);

    // This is primarily for debugging purposes,
    //  to view the pixel values as ascii data.
    //if (true)
    //{
    //    System.IO.StreamWriter file = new System.IO.StreamWriter(
    //                "C:\\imageSigned.txt");

    //    for (int ik = 0; ik < pixels8.Count; ++ik)
    //        file.Write(pixels8[ik] + "   ");

    //    file.Close();
    //}

    minPixelValue = pixels8.Min();
    maxPixelValue = pixels8.Max();


    if (dd.signedImage)
    {
        winCentre -= char.MinValue;
    }

    if (Math.Abs(winWidth) < 0.001)
    {
        winWidth = maxPixelValue - minPixelValue;
    }

    if ((winCentre == 0) ||
        (minPixelValue > winCentre) || (maxPixelValue < winCentre))
    {
        winCentre = (maxPixelValue + minPixelValue) / 2;
    }

    imagePanelControl.SetParameters(ref pixels8, imageWidth, imageHeight,
        winWidth, winCentre, samplesPerPixel, true, this);
}

if (samplesPerPixel == 1 && bitDepth == 16)
{
    pixels16.Clear();
    pixels8.Clear();
    pixels24.Clear();
    dd.GetPixels16(ref pixels16);

    // This is primarily for debugging purposes,
    //  to view the pixel values as ascii data.
    //if (true)
    //{
```

```
                    //    System.IO.StreamWriter file = new System.IO.StreamWriter(
                    //                "C:\\imageSigned.txt");

                    //    for (int ik = 0; ik < pixels16.Count; ++ik)
                    //        file.Write(pixels16[ik] + "  ");

                    //    file.Close();
                    //}

                    minPixelValue = pixels16.Min();
                    maxPixelValue = pixels16.Max();

                    // Bug fix dated 24 Aug 2013 - for proper window/level of signed
images
                    // Thanks to Matias Montroull from Argentina for pointing this out.
                    if (dd.signedImage)
                    {
                        winCentre -= short.MinValue;
                    }

                    if (Math.Abs(winWidth) < 0.001)
                    {
                        winWidth = maxPixelValue - minPixelValue;
                    }

                    if ((winCentre == 0) ||
                        (minPixelValue > winCentre) || (maxPixelValue < winCentre))
                    {
                        winCentre = (maxPixelValue + minPixelValue) / 2;
                    }

                    imagePanelControl.Signed16Image = dd.signedImage;

                    imagePanelControl.SetParameters(ref pixels16, imageWidth,
imageHeight,
                        winWidth, winCentre, true, this);
                }

                if (samplesPerPixel == 3 && bitDepth == 8)
                {
                    // This is an RGB colour image
                    pixels8.Clear();
                    pixels16.Clear();
                    pixels24.Clear();
                    dd.GetPixels24(ref pixels24);

                    // This code segment is primarily for debugging purposes,
                    //    to view the pixel values as ascii data.
                    //if (true)
                    //{
                    //    System.IO.StreamWriter file = new System.IO.StreamWriter(
                    //                        "C:\\image24.txt");

                    //    for (int ik = 0; ik < pixels24.Count; ++ik)
                    //        file.Write(pixels24[ik] + "  ");

                    //    file.Close();
                    //}
```

```
                    imagePanelControl.SetParameters(ref pixels24, imageWidth,
imageHeight,
                        winWidth, winCentre, samplesPerPixel, true, this);
                }
            }
            else
            {
                if (typeOfDicomFile == TypeOfDicomFile.DicomUnknownTransferSyntax)
                {
                    MessageBox.Show("Sorry, I can't read a DICOM file with this Transfer
Syntax.",
                        "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                }
                else
                {
                    MessageBox.Show("Sorry, I can't open this file. " +
                        "This file does not appear to contain a DICOM image.",
                        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }

                Text = "DICOM Image Viewer: ";
                // Show a plain grayscale image instead
                pixels8.Clear();
                pixels16.Clear();
                pixels24.Clear();
                samplesPerPixel = 1;

                imageWidth = imagePanelControl.Width - 25;   // 25 is a magic number
                imageHeight = imagePanelControl.Height - 25; // Same magic number
                int iNoPix = imageWidth * imageHeight;

                for (int i = 0; i < iNoPix; ++i)
                {
                    pixels8.Add(240);// 240 is the grayvalue corresponding to the Control
colour
                }
                winWidth = 256;
                winCentre = 127;
                imagePanelControl.SetParameters(ref pixels8, imageWidth, imageHeight,
                    winWidth, winCentre, samplesPerPixel, true, this);
                imagePanelControl.Invalidate();
                label1.Visible = false;
                label2.Visible = false;
                label3.Visible = false;
                label4.Visible = false;
                bnSave.Enabled = false;
                bnTags.Enabled = false;
                bnResetWL.Enabled = false;
            }
        }

        private void bnTags_Click(object sender, EventArgs e)
        {
            if (imageOpened == true)
            {
                List<string> str = dd.dicomInfo;
```

```
                DicomTagsForm dtg = new DicomTagsForm();
                dtg.SetString(ref str);
                dtg.ShowDialog();

                imagePanelControl.Invalidate();
            }
            else
                MessageBox.Show("Load a DICOM file before viewing tags!", "Information",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        private void bnSave_Click(object sender, EventArgs e)
        {
            if (imageOpened == true)
            {
                SaveFileDialog sfd = new SaveFileDialog();
                sfd.Filter = "PNG Files(*.png)|*.png";

                if (sfd.ShowDialog() == DialogResult.OK)
                    imagePanelControl.SaveImage(sfd.FileName);
            }
            else
                MessageBox.Show("Load a DICOM file before saving!", "Information",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);

            imagePanelControl.Invalidate();
        }

        private void MainForm_Load(object sender, EventArgs e)
        {
            label1.Visible = false;
            label2.Visible = false;
            label3.Visible = false;
            label4.Visible = false;
        }

        private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            pixels8.Clear();
            pixels16.Clear();
            if (imagePanelControl != null) imagePanelControl.Dispose();
        }

        private void bnResetWL_Click(object sender, EventArgs e)
        {
            if ((pixels8.Count > 0) || (pixels16.Count > 0) || (pixels24.Count > 0))
            {
                imagePanelControl.ResetValues();
                if (bitDepth == 8)
                {
                    if (samplesPerPixel == 1)
                        imagePanelControl.SetParameters(ref pixels8, imageWidth,
imageHeight,
                            winWidth, winCentre, samplesPerPixel, false, this);
                    else // samplesPerPixel == 3
                        imagePanelControl.SetParameters(ref pixels24, imageWidth,
imageHeight,
                        winWidth, winCentre, samplesPerPixel, false, this);
```

```
                }

                if (bitDepth == 16)
                    imagePanelControl.SetParameters(ref pixels16, imageWidth,
imageHeight,
                        winWidth, winCentre, false, this);
            }
            else
                MessageBox.Show("Load a DICOM file before resetting!", "Information",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void UpdateWindowLevel(int winWidth, int winCentre, ImageBitsPerPixel bpp)
        {
            int winMin = Convert.ToInt32(winCentre - 0.5 * winWidth);
            int winMax = winMin + winWidth;
            this.windowLevelControl.SetWindowWidthCentre(winMin, winMax, winWidth,
winCentre, bpp, signedImage);
        }

        private void viewSettingsCheckedChanged(object sender, EventArgs e)
        {
            if (rbZoom1_1.Checked)
            {
                imagePanelControl.viewSettings = ViewSettings.Zoom1_1;
            }
            else
            {
                imagePanelControl.viewSettings = ViewSettings.ZoomToFit;
            }

            imagePanelControl.viewSettingsChanged = true;
            imagePanelControl.Invalidate();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
        {
           //this diffenition to display pic from DB and DataGrid to picture Box :(
            BL.CLS_PT getpt = new BL.CLS_PT();
            // here we convert the pictures to Byte ARRAY
           byte[] pic =
(byte[])getpt.GET_IMAGE_PT(this.dataGridView1.CurrentRow.Cells[0].Value.ToString()).Rows[
0][0];
           MemoryStream ms = new MemoryStream(pic); // here we save the array to temp
memory
           picbox1.Image = Image.FromStream(ms);
            // end of pic display code :)

            DataGridViewRow row = this.dataGridView1.Rows[e.RowIndex]; // this code to
fill textboxs from datagrid
            textID.Text = row.Cells["Petiant No"].Value.ToString();
            textPT.Text = row.Cells["Full Name"].Value.ToString();
```

```
            textAGE.Text = row.Cells["Age"].Value.ToString();
            textGEN.Text = row.Cells["Gender"].Value.ToString();

        }

        private void button2_Click(object sender, EventArgs e)
        { //************************************************************

            textID.Text = "";
            textPT.Text = "";
            textAGE.Text = "";
            textGEN.Text = "";
            button2.Enabled = false;

            //@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

        }

        private void textFND_TextChanged(object sender, EventArgs e)
        {

            DataTable dt = new DataTable(); // this to recive incomming data from SQL
            dt = izpt.FND_PT(textFND.Text);
            this.dataGridView1.DataSource = dt;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // First Convert image to byte array.
            byte[] byteArray = new byte[0];
            using (MemoryStream stream = new MemoryStream())
            {
                picbox1.Image.Save(stream, System.Drawing.Imaging.ImageFormat.Png);
                stream.Close();
                byteArray = stream.ToArray();
                izpt.update_PT1(Convert.ToInt32(textID.Text), textPT.Text, textAGE.Text,
textGEN.Text, byteArray);
                MessageBox.Show("PETIANT INFORMATION UPDATED SEUCCESSFULY", "UPDATE",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            }

        }

        private void button3_Click(object sender, EventArgs e)
        {

        }

        private void windowLevelControl_Load(object sender, EventArgs e)
        {

        }




    }
}
```