



Dmitry Rykov

## THREE DIMENSIONAL INTERACTIVE DENTAL CHARTING

# **THREE DIMENSIONAL INTERACTIVE DENTAL CHARTING**

Dmitry Rykov  
Bachelor's thesis  
Spring 2012  
Degree Program in Information Technology  
Oulu University of Applied Sciences

## ABSTRACT

Oulu University of Applied Sciences

Degree Program in Information Technology

---

Author: Dmitry Rykov

Title of thesis: Three Dimensional Interactive Dental Charting

Supervisor: Leo Ilkko

Term and year of completion: Spring 2012

Number of pages: 46 + 6 appendices

---

This Bachelor's thesis presents the development process of a computer aided orthodontic treatment simulation ActiveX control created in Microsoft Visual Studio that demonstrates the usage of the component in the area of a contemporary orthodontic treatment simulation software. Based on an OpenGL application programming interface, this paper will present a graphical user interface system that will demonstrate a three dimensional patient's model of the teeth loaded from a three dimensional object file for a virtual orthodontic treatment system.

To achieve the results an ActiveX technology was chosen to represent the 3D model of the human teeth on the screen because of its simplicity to integrate OpenGL drawing commands.

The user interface design was made with an XML (Extensible Markup Language)-based GUI by which a user can manipulate controls and the 3D scene. Finally, the method was given to define the GUI based on XML configuration files and integration of a 3D teeth model created in a 3DS Max studio and GUI into the ActiveX control. This paper shows that the developed ActiveX 3D orthodontic treatment user interface can be considered as a guideline of treatment preview or a dental charting diagram for patients and dentists. It is also a training tool for dental professionals, orthodontists and students to enhance their experience through the simulation before starting the actual treatment process. Finally, by the end of the development process 3D ActiveX control was successfully tested and fully integrated into a dental practice management software WinHIT.

In the future development this ActiveX control will be enhanced by adding such features as a three-dimensional text over the teeth or over the chosen teeth, i.e. active teeth which are under treatment process at that moment will be displayed. A ray-tracing scene rendering method for a realistic appearance on the screen will be developed. Also, adding a two-dimensional teeth charting diagram would be a supplement for the dentist if he/she wishes so. Such features as loading the model of different formats like Wavefront Object, STL or 3DS will be available in the future releases. Additionally, a DICOM viewer can be integrated to help dental professionals to acquire, monitor and analyze a patient's current state of teeth. A multiple-language support will be developed, too.

---

Keywords: Medical Informatics, Dental Informatics, Dental Models, Three Dimensional Image, Orthodontics, Oral Medicine, ActiveX

## TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu  
Tietotekniikan koulutusohjelma

---

Tekijä: Dmitry Rykov  
Opinnäytetyön nimi: 3D Interaktiivinen dentaalinen katseluohjelma  
Työn ohjaaja: Leo Ilkko  
Työn valmistumislukukausi ja -vuosi: Kevät 2012  
Sivumäärä: 46 + 6 liitteet

---

Tässä opinnäytetyössä oli tavoitteena kehittää graafinen työkalu ortodontian kolmiulotteiseen kuvantamiseen sekä tietokoneavusteiseen simulointiin.

Kolmiulotteisessa kuvantamisessa päädyttiin käyttämään ActiveX-tekniologiaa, koska OpenGL-rajapinnan piirtokomennot ovat siinä hyvin tuettuja. Käyttöliittymän suunnittelussa käytettiin XML-perusteista graafista käyttöliittymää joka mahdollisti kontrollien sekä kolmiulotteisen näkymän yksinkertaisen muokkaamisen.

Työn tuloksena toteutettiin kolmiulotteinen hampaiston ActiveX-katseluohjelma, joka hyödyntää OpenGL-ohjelmistorajapintaa. Katseluohjelmaa voidaan hyödyntää toteutuneen ortodontisen hoidon seurannassa. Lisäksi työn tuloksena saatua ortodontian tietokoneavusteista simulointia voidaan hyödyntää päätöksenteon tukena hammaslääkäreiden, oikomishoitajien sekä alan opiskelijoiden keskuudessa.

Työn tuloksena saatu kolmiulotteinen hampaiston ActiveX-katseluohjelma integroitiin ja testattiin onnistuneesti suun terveydenhuollon WinHIT-tietojärjestelmässä.

Jatkokehityksessä tähän ActiveX-komponenttiin tullaan lisäämään ominaisuuksia, esim. kolmiulotteista tekstiä hoidon kohteena olevan hampaan kuvan päälle. Toinen ominaisuus voisi olla säteen seurannan mallinnusmenetelmä näytölle. Myös kaksiulotteinen hammaskaaviokuva voidaan lisätä, jos hammaslääkäri niin haluaa. Seuraavaan versioon voidaan sisällyttää vielä lisää ominaisuuksia, esim. eri formaattien, kuten Wavefront Object:n, STL:n ja 3DS:n latausmalli. Lisäksi tähän sovellukseen voidaan integroida DICOM-katseluohjelma auttamaan hammashoidon ammattilaisia potilaan hampaiden nykytilan tutkinnassa ja analysoinnissa. Ohjelman eri kieliversioita kehitetään myös.

---

Asiasanat: Lääketieteellinen informatiikka, Hammaslääketieteellinen informatiikka, Hampaistomallit, Kolmiulotteinen kuvantaminen, Ortodontia, Suulääketiede, ActiveX

## PREFACE

The development process was accomplished during the winter and spring term of the year 2011/2012 for In Net Oy company, which is located in Softpolis of Raahe city. The company specializes in developing the dental software solutions for dental companies, oral healthcare centers and dental practitioners in Finland. This thesis primarily focuses on the development of the interactive ActiveX 3D dental charting for the WinHIT dental practice management software.

I want to thank my supervisor Leo Ilkko and the company In Net Oy and personally Jari Moisanen and the head of In Net Oy company Esko Ristkari for the great opportunity to let me complete my thesis for the company. I would also like to express best wishes and sincere appreciation to my supervisor Leo Ilkko for his incredible tutoring and organizing skills and my colleagues for the willingness to help and give advice.

# CONTENTS

<b>PREFACE</b>	5
<b>CONTENTS</b>	6
<b>SYMBOLS AND ABBREVIATIONS</b>	8
<b>1. INTRODUCTION</b>	9
1.1 Research problems and methods	11
1.2 Iterative and incremental development methodology	13
<b>2. WORKING ENVIRONMENT</b>	14
<b>3. DEFINITION</b>	16
3.1 User's perception	18
3.2 Technologies	20
3.2.1 ActiveX technology	20
3.2.2 OpenGL Graphics System	21
3.2.3 OpenGL in MFC ActiveX component	23
3.2.4 XML based GUI	24
3.2.5 3D teeth model design	25
3.2.6 3DS model in 3DS Max	26
3.2.7 3D object formats	27
<b>4. IMPLEMENTATION</b>	30
4.1 Architecture	30
4.2 ActiveX control	32

4.3 3D-Teeth Model	34
4.4 XML based user interface	36
4.5 UI Screen	37
4.5.1 Teeth model rendering	37
4.5.2 Teeth model manipulation	39
4.5.3 XML UI Settings	39
<b>5. TESTING</b>	<b>40</b>
<b>6. FUTURE DEVELOPMENT</b>	<b>41</b>
6.1 Functionality	41
6.2 DICOM Viewer	42
6.3 UI Settings	43
6.4 Ray-tracing rendering	43
6.5 Teeth Charting	44
<b>7. CONCLUSION AND RESULTS</b>	<b>45</b>
<b>8. DISCUSSION</b>	<b>46</b>
<b>9. REFERENCES</b>	<b>47</b>
<b>10. APPENDICES</b>	<b>49</b>

## SYMBOLS AND ABBREVIATIONS

ActiveX	Framework for defining reusable software components
API	Application Programming Interface
CAD	Computer Aided Design
CAM	Computer Aided Modeling/Manufacturing
COM	Component Object Model
DICOM	Digital Imaging and Communications in Medicine
FDI	Fédération Dentaire Internationale
GUI	Graphical User Interface
IDE	Integrated Development Environment
ISO	International Organization for Standardization
LCD	Liquid Crystal Display
MFC	Microsoft Foundation Class Library
MTL	Material Template Library format
OBJ	Wavefront 3D Object File Format
OCX	Object Linking and Embedding Control Extension
OLE	Object Linking and Embedding
OpenGL	Open Graphics Library
OS	Operating System
STL	Stereo Lithography File
TFT	Thin-Film Transistor
UI	User Interface
UML	Unified Modeling Language
WinHIT	Windows Hammashoidon Integroitu Tietojärjestelmä
WWW	World Wide Web
XML	Extensible Markup Language
3D	Three dimensional
3DS	Three Dimensional Studio File
3DS Max	Three Dimensional Studio Max



## 1. Introduction

In the past decades user interfaces and eventually all teeth charts for dental management practice software were simply two dimensional charts and they represented just a bunch of interconnected tables and simple graphs to make it easier for dental professionals to handle the list of appointments and conduct dental treatment procedures. But since the introduction of 3D API libraries, powerful processors and video cards for the ordinary desktops the implementation of contemporary user interfaces were presented. Dental software development companies realized the growing potential of new contemporary user interfaces with 3D graphics.

In order to create some innovative and modern dental charting interface for the WinHIT dental practice management software, which is produced by In Net Oy company located in Softpolis of Raahe. (In Net Oy 2012, date of retrieval 12.02.2012), the ActiveX interactive dental charting component required to be developed to represent the 3D model of the teeth in the form of a dental chart. This component represents the dental charting interactive interface in order to provide a complete and reliable way to represent the three-dimensional model of the human teeth on the computer display.

To develop this component an ActiveX technology was chosen for the reason of the compatibility with the existing software and application programming interfaces to make integration seamless and painless with minimal efforts. (Microsoft 2012, date of retrieval 11.02.2012) ActiveX together with another technology OpenGL API allows creating an incredibly natural appearance of the human teeth without virtual reality simulators or ray-tracing rendering methods.

During the typical process of patient's teeth condition check-up, the dentist or his assistant uses the ordinary two-dimensional interactive teeth charting interface of the WinHIT dental practice management software to plan the treatment process of problem teeth and update the dental records of the patient's health condition after the check-up procedure into the database. On the other hand, to simplify the communication between a dental professional and his assistant the same teeth name conventions are used. (American Dental Association 2010, date of retrieval 12.02.2012)

Despite the convenience, this approach has a lot of drawbacks, especially the lack of understanding the process between the dentist and the assistant and the lack of clarity and simplicity. So the new approach is required by the modern dental healthcare. Meanwhile switching from two-dimensional or paper-based approach into a three-dimensional one requires quick-to-react, robust and user-friendly interfaces to make simple and comprehensive interaction between the dentist and the dental treatment simulation software.

Primarily, this software development process focuses on implementing a new way of representation of a teeth model to make the dental treatment process more comprehensive, and to increase the interactivity between the different parts of the treatment process including dentist and assistant communication and patient-to-dentist communication as well. This simplified way of representing the patient's teeth as a 3D model rendered in ActiveX component can be described as patient's teeth map, i.e. a dental chart. (FDI World Dental Federation 2012, date of retrieval 5.02.2012) After all, a higher level of interactivity is achieved through the usage of touch-screen TFT-LCD displays at dentist's workplace.

## 1.1 Research problems and methods

After the task definition, the research problems must be described and explained as required. So during the process of an ActiveX interactive dental charting control development and implementation process the following things had to be done:

1. Developing a 3D model of the human teeth based on a convenient teeth charting standard
2. Developing the XML based GUI interface to load and manipulate the 3D teeth model
3. Integrating of the GUI and 3D teeth model into the MFC ActiveX control
4. Deploying and testing the ActiveX control on a target Windows OS platform

The 3D model was developed by using 3DS Studio Max. The Right Hemisphere modeling software tool was used to design and test the color materials of the teeth and gums. In this development process of the 3D teeth model, the above mentioned applications were used as primary tools for designing the teeth, gums, enamel and the internal structure of the human jaw.

Visual C++ was chosen as a primary development object-oriented programming language to develop an MFC ActiveX control for the Windows platform, and an XML based GUI was integrated in order to simplify operations conducted by the user.

OpenGL was chosen as the main 3D graphics API to implement the rendering of the GUI and 3D model in the development process of the ActiveX dental charting control. OpenGL has a lot of advantages compared to its competitors in this field.

OpenGL is the industry's most widely used, supported and best documented 2D/3D graphics API making it inexpensive and easy to obtain information on implementing OpenGL in hardware and software. There are numerous books, tutorials, online coding examples, coding seminars, and classes that document the API, Extensions, Utility Libraries, and Platform Specific Implementations. (The Khronos Group 2012, date of retrieval 5.02.2012)

OpenGL is supported on every major operating system, it works with every major windowing system, and it is callable from most programming languages. It offers complete independence from network

protocols and topologies. All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system. (The Khronos Group 2012, date of retrieval 5.02.2012)

The development and research methods are primarily based on reading, exploring and analyzing the technical literature, specifications, books, manuals, tutorials on above-mentioned topics concerning the usage of the tools, creating the reusable software components, user interface development, medical conferences, programming guides, 3d-object modeling and all other kinds of activities in order to analyze, gather and combine all of them together to achieve the maximum efficiency, performance and robustness of the final software product. Following all these steps in the research and development process, the chosen research method became the main tool to solve all problems that appeared during all four development stages.

Books, programming guides, research papers, technical literature, available software reviews and medical products were used as the sources of information during the research and development process of the ActiveX control and writing the bachelor's thesis.

## 1.2 Iterative and incremental development methodology

During the development process of the 3D ActiveX dental charting control the iterative and incremental agile software development methodology was used in order to meet the requirements of the software product and successfully build, test and deploy the product on the target system. The iterative and incremental development can be described as the successor of a waterfall development method due to its weaknesses. It was created in response to the issues and weaknesses of the waterfall development approach. This development process starts with an initial planning of the development and ends up with cyclic interactions between them. Figure 1 shows this process of development graphically.



FIGURE 1. Iterative and incremental development.

In an iterative and incremental lifecycle (Figure 1), the development proceeds as a series of iterations that evolve into the final system. Every iteration consists of the following process components: requirements analysis, analysis, design, implementation, and test. The developers do not assume that all requirements are known at the beginning of the lifecycle, indeed change is anticipated throughout all phases. (Cockburn 2008, date of retrieval 19.03.2012)

## 2. WORKING ENVIRONMENT

In order to complete all stages of the development process and develop the final product to meet the defined and described requirements, Microsoft Visual Studio C++ were used as the main working environment IDE together with the latest OpenGL 4.2 core specifications (The Khronos Group 2012, date of retrieval 6.02.2012). Visual Studio is considered to be the best working environment for the Windows platform. It is comprehensive, easy to use, user-friendly, well documented and very fast. This is the best approach to develop, maintain, debug, test and deploy your future applications on the Windows platform. The IDE provides all needed libraries, header files to produce the source code of the project.

The Autodesk 3DS Max Studio was chosen as the development environment for the 3D teeth model creation and editing process.

Autodesk 3DS Max and Autodesk 3DS Max Design software provide powerful, integrated 3D modeling, animation, rendering, and compositing tools that enable artists and designers to more quickly ramp up for production. The two versions share core technology and features, but offer differentiated experiences and specialized toolsets for game developers, visual effects artists, and graphic designers on the one hand, and architects, designers, engineers, and visualization specialists on the other. (Autodesk 2012, date of retrieval 6.02.2012)

The Right Hemisphere Deep Exploration 5.7 was used as the main tool in editing the 3D teeth model and to convert the space coordinates of the 3D teeth model into a C++ code and after that the teeth model was loaded from the memory. Also, this program was very useful in creating the teeth object's materials and handling a 3D-object structure and groupings in a 3D scene.

With Deep Exploration, existing CAD models are transformed into ultra-lightweight files ready to use in the authoring and publishing of 2D and 3D product graphics, a wide variety of documents, procedures and training curriculum. All these capabilities are supported from a single CAD model utilized within Deep Exploration. (Right Hemisphere 2010, date of retrieval 7.02.2012)

ActiveX Control Test Container was used as a primary debugging tool for the purposes of loading the control with the 3D teeth model and graphical user interface in it and invoking the ActiveX control's methods and firing events.

ActiveX Control Test Container can display logs of events and data-binding notifications. It also provides facilities to test a control's persistence by letting you save properties to a property bag, to a stream, or to a storage. You can also use Visual Basic Scripting Edition to automate your use of ActiveX Control Test Container. You can modify the default features that the container implements. You can also select which optional features the container implements. For example, you can specify that the container supports windowless controls. This ensures that your control will work in a variety of containers. For additional information about using ActiveX Control Test Container, including information on its user interface, use the help file that accompanies the test container. You can open help from the Help menu, and by using the context sensitive help (pressing the F1 key) in the user interface. (Microsoft 2012, date of retrieval 7.02.2012)

To specify, standardize and define the structure of the software system's design, the UML diagram modeling tool is essential. For creating and editing the development workflow of the software project architecture and modeling UML diagrams, the StarUML was chosen for those purposes.

### 3. DEFINITION

The developing system can be described as the software, an ActiveX control, which acts as an interactive dental charting three-dimensional scene with human teeth in it, based on OpenGL API and from the point of view of the programmer, it acts like a frame control. This software system incorporates modern OpenGL techniques to produce a real-life appearance of the human teeth, to make the user's activities simpler and to display the user interface controls in a straight-forward manner.

The three-dimensional graphical user interface ActiveX control, which is based on OpenGL and XML technologies, will later be referred simply as "the application". It is a software application, which will make the process of the dental clinical planning treatment faster and easier. The results will be handled in the report planning activities by the dentist or his assistant in a more clear way when the treatment results are presented to another dental professional or straight to the patient.

In order to support multiple Windows OS versions, all modern video cards and video drivers the OpenGL technology was chosen. The main reason for this was the key factor of its best suitability to meet the requirements of compatibility with other platforms, operating system versions, video card's drivers and the current software architecture of WinHIT dental practice management software. OpenGL fully meets the above-mentioned requirements, so the choice of this technology shows and explains the major issues that were resolved during the development process.

In the development process the WinHIT can be considered as the "host" application where the 3D ActiveX control to be developed will be integrated to switch the treatment planning from an outdated 2D-dental chart to a new developed 3D-dental chart. WinHIT is a dental practice management software, which is the main product developed by In Net Oy company. According to Gruber, Pollack and Rosenthal (1994, 166-171), a clinical practice software is considered to bring a lot of improvements to the relationship between the dentists and the patients in a new way and it can improve the healthcare services that are available to the patients through the benefits of the new accessible technologies.



This application will be responsible for loading the 3D-space coordinates of the human teeth model in the right form into the computer's dynamic memory, render them to geometrical primitives to produce a complete realistic appearance of the human jaws on the display. The application can be considered as a replacement of the old two-dimensional teeth charting ActiveX control to represent the human teeth before and after the treatment procedures made by the dentist (see Figure 2).

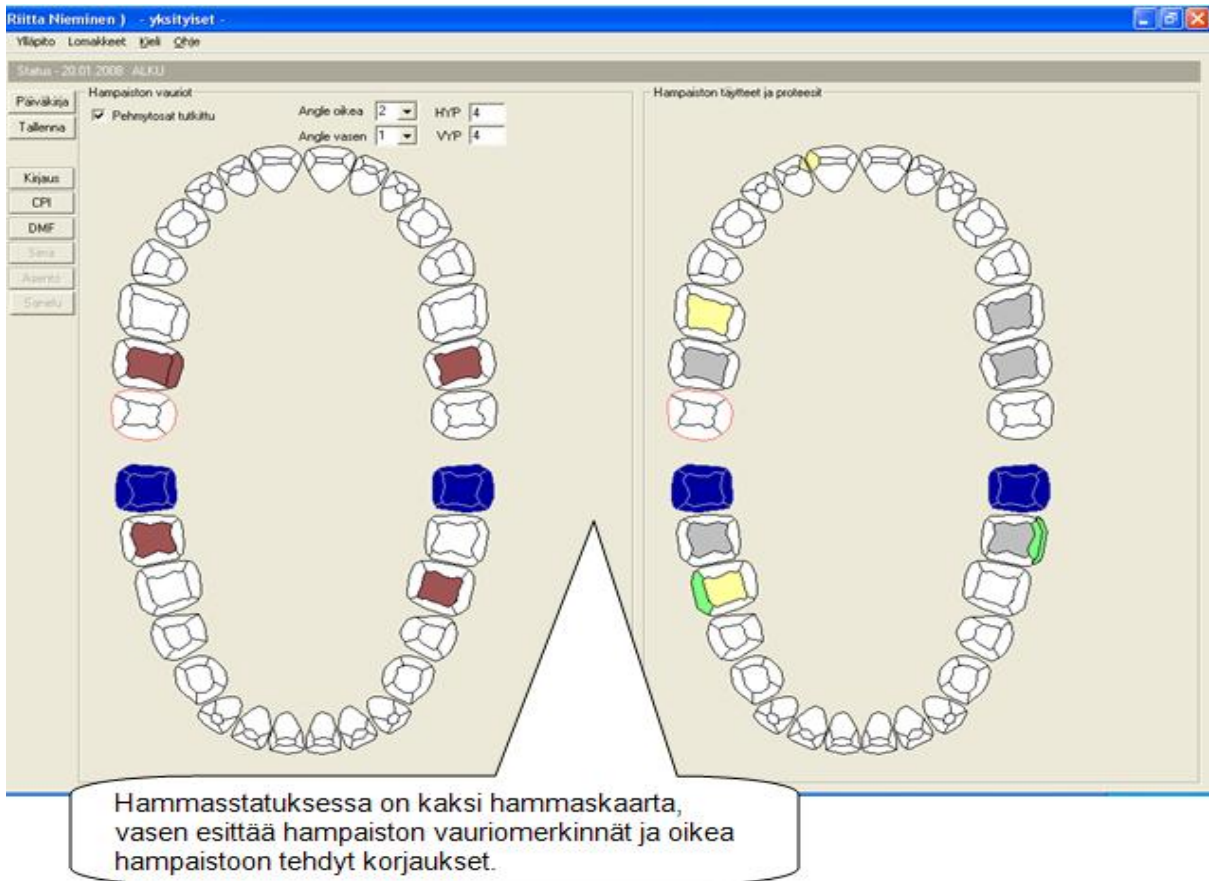


FIGURE 2. 2D interactive teeth charting ActiveX control in WinHIT treatment planning process.

The development process will be split into three main stages:

- Developing the 3D model of the teeth to represent dental charting
- Designing the graphical user interface based on XML description files
- Combining previous stages to integrate those components into the ActiveX component

### 3.1 User's perception

In order to create an application that will be easily used by the dentist and the assistant, the application's user interface and 3D model teeth charting notation and its numbering needs to be based on the standard layout which is considered to be the common standard in the dental software management practice. There are three major standards of dental charting notation existing in the world today:

- FDI World Dental Federation notation
- Universal Numbering notation
- Palmer notation

The first notation, FDI World Federation notation, is the wide-spread notation that is used today. This notation is also known as the FDI Two digit notation or the ISO 3950 notation (Hussain, Kachymalay & Aziz 2010, 794-797). This charting notation is widely used in Finland by local dentists and health care centers (see Figure 3).

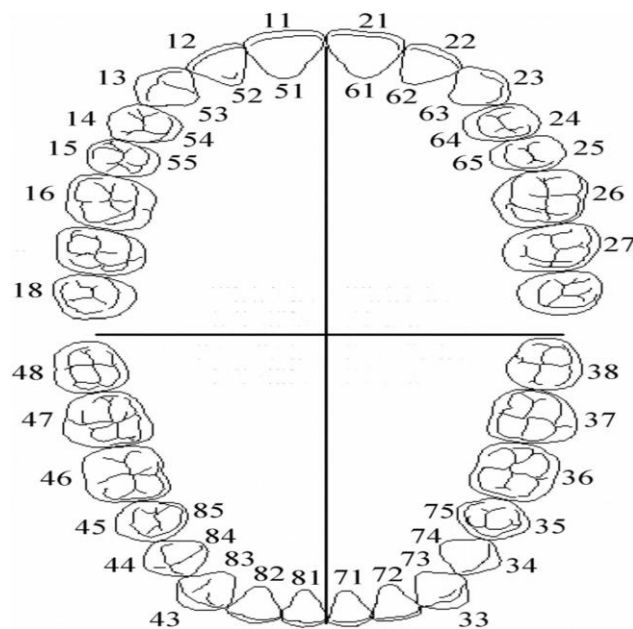


FIGURE 3. The FDI teeth numbering for primary and permanent teeth.

The first digit represents the patient's mouth quadrant, symmetrically numbered from number 1 to number 4, starting clockwise, from the upper-right. The second digit in this FDI dental notation chart indicates the tooth type in each quadrant displayed on the chart, numbered from 1 to 8, from the center molar and moving backwards. This notation is used by the World Health Organization. (Hussain et al. 2010, 794-797)

The second notation, which can also be found as a standard teeth notation in other countries, is called the Universal Numbering notation or the National System. This notation is used in the United States. The patient's teeth are numbered from 1 – upper right third molar to 16 – upper left third molar following around the upper jaw and going down to the lower jaw, from number 17 to the number 32. (Hussain et al. 2010, 794-797)

The third notation is called the Palmer notation. It is primarily used in the UK. This notation consists of four symbols (┘ ┌ └ ┐) representing the quadrant in which the specific tooth can be found, indicating the position from the middle of the jaw. Permanent (adult) teeth numbered from number 1 to 8 and deciduous teeth described by letters from A to E in an alphabetical order. According to the studies conducted by Blinkhorn, Choi & Paget (1998, 39-41) the Palmer notation is still more often used in the practice by British dentists than in other countries.

The application's user interface developed in this thesis presents the natural way of handling the 3D-teeth model by the dentist. And the usage of the interface will be easy and quite friendly to the dental professional who does not have any previous experience dealing with a 3D-teeth model as dental practice management software. The appearance of the application, developed during the thesis work, will appear in the form of the dental simulator, so the realistic look will make the dental treatment procedures and results to be presented in a quite easy-to-comprehend manner. (Marotta, Phanichphant, Malatack, Shah, Price, Thyvalikakath, Schleyer & Hong 2007, 2567-2572) In the work environment of every dentist every modification made by the dentist in the treatment plan will affect the records concerning the specific patient in the database. The teeth numbers and surfaces (numbered from 1-5) will be mapped into the Titanium database which is used by In Net Oy to store the patient's detail. But it is not the part of the thesis work. So the details concerning the database structure will be omitted.

## **3.2 Technologies**

### **3.2.1 ActiveX technology**

ActiveX is a bunch of technologies developed by Microsoft Corporation in the 1990's. These technologies are based on COM (Component Object Model) architecture and eventually all of them are nowadays referred and related to the variety of COM-based technologies. (Chappell 1996, 4-7)

ActiveX is a component program object that can be reused many times by the other software components and also by many existing applications. The Component Object Model is an umbrella for the other different ActiveX technologies gathered into the single core technology.

Basically, an ActiveX control is not a programming language or a library either, but a set of descriptive and comprehensive rules about how two different software components should communicate with each other using the same rules, formats and interfaces. It was originally called OLE (Object Linking and Embedding) control, which uses an OCX file extension in a Windows environment. The OCX control is a software component, a frame and container to embed in it a piece of software that needs to be executed by the developer. This component can encapsulate 3D drawing functions, libraries and interfaces. ActiveX allows the existing software components to function without rewriting the code from the scratch. So in that way the development process becomes much simpler. So the developers can create their software components ranging from simple demo scenes, CAD/CAM drawing controls to interactive demos and so on. (Min & Xiaoyan 2007, 559-560)

Due to the above-mentioned properties and advantages of the ActiveX technology, it was used as the core technology that is needed to get acquainted in more detail before the beginning of the development process. The existing WinHIT's program interfaces also required the ActiveX control to be fully compatible with the existing software. And this is the reason why the ActiveX technology was chosen for the implementation of this ActiveX interactive dental charting control in this thesis.

### 3.2.2 OpenGL Graphics System

The OpenGL Graphics System can be described according to the Architecture Review Board of OpenGL as follows:

OpenGL (for “Open Graphics Library”) is a software interface to graphics hardware. The interface consists of a set of several hundred procedures and functions that allow a programmer to specify the objects and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects. Most of OpenGL requires that the graphics hardware contain a framebuffer. Many OpenGL calls pertain to drawing objects such as points, lines, polygons, and bitmaps, but the way that some of this drawing occurs (such as when antialiasing or texturing is enabled) relies on the existence of a framebuffer. Further, some of OpenGL is specifically concerned with framebuffer manipulation. (Segal & Akeley 2006, 1-2)

From the point of view of a programmer, the OpenGL graphics system can be described as follows:

To the programmer, OpenGL is a set of commands that allow the specification of geometric objects in two or three dimensions, together with commands that control how these objects are rendered into the framebuffer. For the most part, OpenGL provides an immediate-mode interface, meaning that specifying an object causes it to be drawn. A typical program that uses OpenGL begins with calls to open a window into the framebuffer into which the program will draw. Then, calls are made to allocate a GL context and associate it with the window. Once a GL context is allocated, the programmer is free to issue OpenGL commands. Some calls are used to draw simple geometric objects (i.e. points, line segments, and polygons), while others affect the rendering of these primitives including how they are lit or colored and how they are mapped from the user’s two- or three-dimensional model space to the two-dimensional screen. There are also calls to effect direct control of the framebuffer, such as reading and writing pixels. (Segal et al. 2006, 2)

From the implementer’s point of view, the OpenGL graphics system can be described as follows:

To the implementor, OpenGL is a set of commands that affect the operation of graphics hardware. If the hardware consists only of an addressable framebuffer, then OpenGL must be implemented almost entirely on the host CPU. More typically, the graphics hardware may comprise varying degrees of graphics acceleration, from a raster subsystem capable of rendering two-dimensional lines and polygons to sophisticated floating-point processors capable of transforming and computing on geometric data. The OpenGL implementor’s task is to provide the CPU software interface while dividing the work for each OpenGL command between the CPU and the graphics hardware. This division must be tailored to the available graphics hardware to obtain optimum performance in carrying out OpenGL calls. OpenGL maintains a considerable amount of state information. This state controls how objects are drawn into the framebuffer. Some of this state is directly available to the user: he or she can make calls to obtain its value. Some of it, however, is visible only by the effect it has on what is drawn. One of the main goals of this specification is to make OpenGL state

information explicit, to elucidate how it changes, and to indicate what its effects are. (Segal et al. 2006, 2)

From the point of view of the creators of the OpenGL standard, Mark Segal and Kurt Akeley, this standard can be defined as follows:

We view OpenGL as a state machine that controls a set of specific drawing operations. This model should engender a specification that satisfies the needs of both programmers and implementors. It does not, however, necessarily provide a model for implementation. An implementation must produce results conforming to those produced by the specified methods, but there may be ways to carry out a particular computation that are more efficient than the one specified. (Segal et al. 2006, 3)

The basic operations of the OpenGL state machine can be described as follows:

OpenGL (henceforth, the “GL”) is concerned only with rendering into a framebuffer (and reading values stored in that framebuffer). There is no support for other peripherals sometimes associated with graphics hardware, such as mice and keyboards. Programmers must rely on other mechanisms to obtain user input. The GL draws primitives subject to a number of selectable modes. Each primitive is a point, line segment, polygon, or pixel rectangle. Each mode may be changed independently; the setting of one does not affect the settings of others (although many modes may interact to determine what eventually ends up in the framebuffer). Modes are set, primitives specified, and other GL operations described by sending commands in the form of function or procedure calls. Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of an edge, or a corner of a polygon where two edges meet. Data (consisting of positional coordinates, colors, normals, and texture coordinates) are associated with a vertex and each vertex is processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that the indicated primitive fits within a specified region; in this case vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents. Commands are always processed in the order in which they are received, although there may be an indeterminate delay before the effects of a command are realized. This means, for example, that one primitive must be drawn completely before any subsequent one can affect the framebuffer. It also means that queries and pixel read operations return state consistent with complete execution of all previously invoked GL commands, except where explicitly specified otherwise. The GL provides direct control over the fundamental operations of 3D and 2D graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. It does not provide a means for describing or modeling complex geometric objects. Another way to describe this situation is to say that the GL provides mechanisms to describe how complex geometric objects are to be rendered rather than mechanisms to describe the complex objects themselves. (Segal et al. 2006, 4)

### 3.2.3 OpenGL in MFC ActiveX component

In order to display the graphical user interface and the 3D-teeth model, the rendering context needed to be created to enable rendering the 3D scene by OpenGL drawing commands. So an MFC ActiveX needs to be defined to make the control that will render the graphics. The Figure 4 describes the interactions between the container of the ActiveX control, which is the window of the application. It will load a newly developed ActiveX control or it can be the ActiveX Control test container tool, which is available in a standard tool repository on any installed Windows OS and the ActiveX control itself.

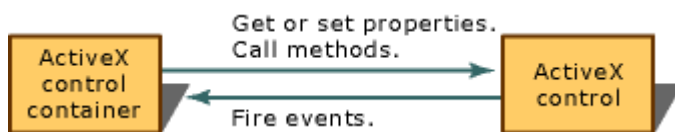


FIGURE 4. Interaction between an ActiveX Control Container and a Windowed ActiveX Control.

Another advantage of having the ActiveX technology is that it is available in the broad spectrum of Microsoft products and can be simply integrated into the existing MFC OpenGL projects. So it is not necessary to reinvent the wheel and begin coding from the scratch. A lot of documentation is available that describes the usage and naming conventions of the OpenGL drawing commands in the MFC application. (Microsoft 2006, date of retrieval 12.03.2012) So according to that guide (Microsoft, 2006) all that is necessary is just to include the gl.h, wgl.h and glu.h header files in order to access the function declarations of the OpenGL application programming interface. Gl.h contains all core functions of the OpenGL API and glu.h contains all utility functions that will help to solve minor problems during the implementation phase of some piece of functionality. So in order to setup the OpenGL drawing context and initialize all the necessary OpenGL state machine variables, an OnCreate() function is necessary. The OpenGL rendering code can be placed in the OnDraw() function to enable a 3D rendering as required. (Microsoft 2012, date of retrieval 12.02.2012) This code can also be placed in the OnTimer() function. This function is suitable for the code that is necessary to be executed when the operating system gives the execution time and priority to your application. The choice was justified due to its simplicity of integration with the OpenGL technology due to the fact that the OpenGL is the most popular technology in the area of dental imaging, computer graphics modeling, design and visualization.

### 3.2.4 XML based GUI

In order to provide the dental care using computer software for the dental professional or for their assistant, the easy-to-use user graphical interface needs to be developed and introduced. So the users would have the operation of handling the three-dimensional model of the teeth more convenient and comprehensive. An XML based technology was chosen because the simplicity, accessible documentation, developer guides and the speed of development made the decision obvious from the point of view of the programmer.

According to the official description of the XML standard from World Wide Web Consortium, this technology can be described as a simple and very flexible text format to represent and exchange data in an electronic form, but originally it was designed to meet the challenges of large-scale electronic publishing in the web. (W3C 2003, date of retrieval 14.03.2012) Therefore, the features of this technology were needed to develop a very flexible and portable graphical user interface in order to maintain the compatibility between different software platforms, software systems, operating systems and libraries that were used in the development process.

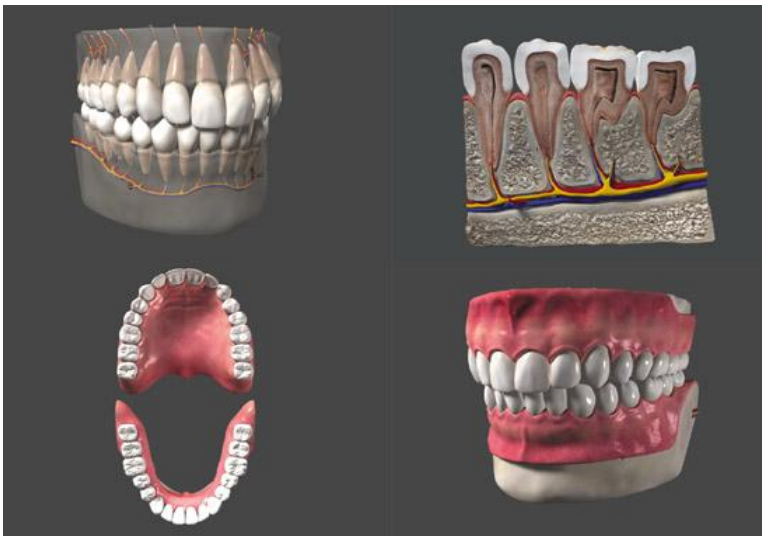
The XML technology will provide the textual description and representation of what should be displayed on the monitor for the dentist so the interface can easily be set by the dentist according to the individual favor of every dentist or their assistant, if they wish so. So the main idea behind this is to make the whole architecture very simple in order to shorten the development time of the user interface and improve the quality of the object-oriented design of the system. According to Chen & Zheng (2008, 381-382) the basic XML configuration file will contain the nodes that describe the layout, design and parameters to represent the parameters for the GUI classes that are implemented using C++ and will load this configuration file to draw the user interface controls. So this XML configuration file is created as a roadmap for the drawing OpenGL subsystem of how this user interface control should look like. Considering all that is described above, the interface definition for XML configuration files were introduced. All user interface controls together with the 3D-teeth model will be displayed inside the MFC ActiveX control that was defined in the preceding subchapter.



### 3.2.5 3D-Teeth model design

In order to represent human teeth on the computer screen the creation and design of the 3D model in the 3D CAD/CAM software were necessary. This model will demonstrate the human teeth based on the two-digit dental charting notation (Hussain et al 2010, 795). Basically, the 3D model can be treated as a mesh of interconnected points in a 3D-coordinate space connected by the lines that form the polygons. Then the textures or color material is applied to those polygons to make the appearance of the model more natural. Shading, multiple light sources and smoothing are also applied to the teeth model to make a real-life appearance on the screen.

The design of this teeth model is justified due to the fact that those kinds of models are necessary for the dental treatment simulation which has become very popular nowadays because of the availability of the tomography devices and 3D-dental scanners on the market (Sinthanayothin & Tharanont 2008, 81-84). The 3DS Max software from Autodesk was used during the development process of the 3D-teeth model. From the point of view of a dental professional, this model is the representation of the previous 2D model in a 3D space and also, the transition from a 2D-view to a 3D-view to display realistic teeth like those shown on the Figure 5 below.



*FIGURE 5. 3D-teeth representation.*

### 3.2.6 3D model in 3DS Max

Nowadays, when there is a need to create realistic 3D model objects, demo scenes and 3D models of professional quality, a film quality animation or a real-life physical phenomenon simulation - 3DS Max is a silver bullet. The 3DS Max software from an American company Autodesk is used as the most popular software tool for accomplishing those kinds of things.

Everything that is created in the 3DS Max can be viewed and located in a three-dimensional world. The 3DS Max has features for viewing an enormous 3D stage from the perspective of the smallest object to the full extent of the development scene. The 3DS Max was chosen because of its incredible features that were necessary during the creation of the 3D-teeth model. It allows creating, manipulating, editing and applying materials to your 3D model (Autodesk 2007, 1-8). During the development of the 3D model for this bachelor's thesis, the version 9 was used as the main working environment for designing and modeling purposes. On the Figure 6 below you can see the 3DS Max main window and a 3D-development scene with a 3D mesh of human teeth in it.

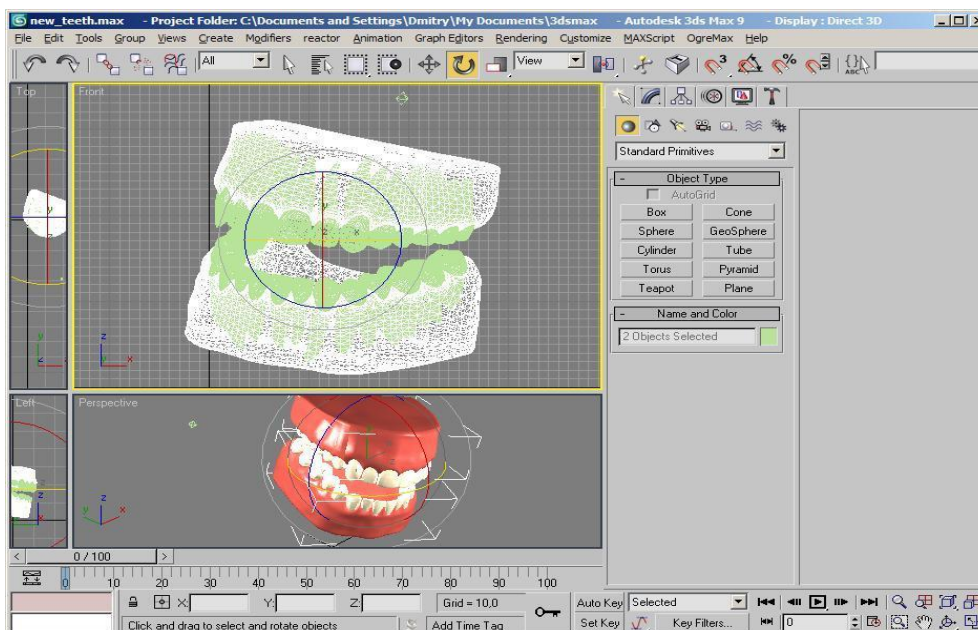


FIGURE 6. Developing 3D teeth model in 3DS Max 9.

### 3.2.7 3D object formats

Everything that is created in a 3DS Max studio is an object that can be stored in different kinds of formats for exchanging a 3D-object's data with another CAD/CAM software or to convert an object's data into a different format to make it more simpler to handle the object's coordinates and other information concerning the 3D scene, such as light sources, object materials, object colors, textures and so on. The default object format to store the modeling object is a proprietary .max format. All information is saved to the file with .max extension. During the development phase of the 3D dental charting ActiveX component development other formats were also explored and studied in order to use them for development purposes.

The list of 3D-object formats that were used in the development of this project is as follows:

- 3D MAX (.MAX)
- 3DS Max format (.3DS)
- Wavefront Object (.OBJ)

3D MAX is a default format that is used by the 3DS Max when you create your first 3D model. Unfortunately, the specification of this format is not public. That was one of the reasons why during the development phase of this project, it was necessary to convert a 3D-teeth model stored in a 3D MAX format into another public format that can be converted easily into other formats or even directly into the C++ code, so that 3D teeth model's coordinates could be easily manipulated during the development phase.

The next most suitable format was 3DS Max (designated file extension .3ds) due to its simplicity in development process and the ability to open this file format with other 3D modeling tools. Shortly, the 3DS format can be described as the file object that consists of the chunks, i.e. small blocks of information that describe the internal hierarchy of the file. 3DS is a binary format that is faster to parse and load by a loader which can be implemented using any available programming language, for example C++ or any other, that is suitable in the specific situation. 3DS format is based on the

chunks where every section of data field is integrated into the block of information, which contains a chunk identifier and according to that id, also the length of the data and also the data in it. These chunks represent a hierarchical structure which is similar to a binary tree structure. This means that each chunk of the data block can be regarded as a node. So we can represent each node as an object of the tree data structure (Cormen, Leiserson, Rivest & Stein 2009, 246-249). But considering the shortcomings of that format, such as a poor documentation and an undocumented hierarchy of the chunks and other proprietary, non-disclosure information makes the development of the loading, sorting and handling algorithm very difficult to implement as required. Despite the popularity of this format, it is not suitable for exchanging the 3D object data in the development process due to the reasons of missing information concerning any developments or any other information that describes how to work with this format from the program written in C++. That's why the Wavefront .OBJ file format was chosen.

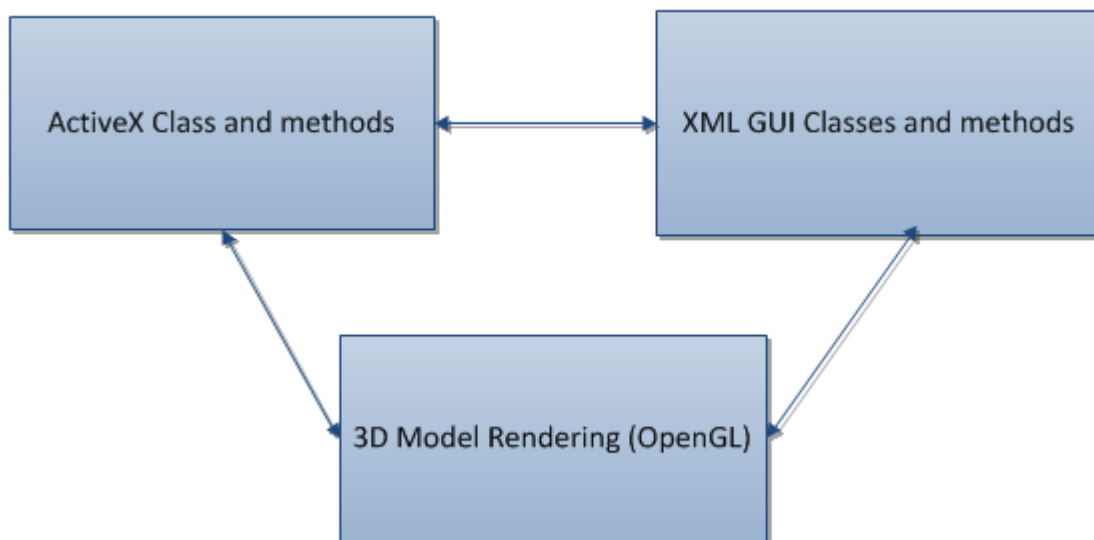
According to the specification which is given by Jones and Sherrod (2011, 342-343), a Wavefront Object file format is just a text-readable ASCII format that can be opened by any text editor. It is very simple to read and get the 3D information concerning the 3D object structure of the modeling scene object. The format is very simple and clear, and its layout is based on the description of the vertices of the 3D object followed by texture coordinates, vertex normal vectors and triangle indices. There is a separate line with a different type of information concerning the description of your 3D model. One of the most important information of your 3D object is vertices coordinates and position in the space. Vertex information starts with a v keyword in .obj file and X, Y and Z coordinates are described after each other in this order and separated by a whitespace so the vertex information can be set or read. Texture coordinates begin with a vt keyword in .obj file and they have floating-point values that follow the keyword and normals start with a vn keyword. The information concerning triangle coordinates starts with an f keyword. After this keyword, there are three groups of values. Every group has the information about the position of the vertex index in a vertex list, a texture coordinate and a normal specified in the .obj file, which has separate indices for every attribute. There are also other keywords in the .obj file used by the mesh handling library, like usemtl. This keyword tells to the .obj loading library that the usage of the color materials is presented in this 3D model.

So that was all about the .obj internal structure. This file format was used throughout the development process in order to store the coordinates of the 3D-teeth model for the purpose of loading the coordinates straight from the memory and thus, avoiding reading them all the time from the disk, and to eliminate all delays in a 3D-rendering scene and to increase the overall performance of the whole 3D ActiveX component. That is necessary when developing the 3D scene with a hundred of thousands of triangles in order to make appearance of the model more realistic.

## 4. IMPLEMENTATION

### 4.1 Architecture

In this chapter the process of the implementation of the 3D interactive dental charting ActiveX component is described in details. The project consists of different modules i.e. classes. There are following classes that create the whole architecture of the ActiveX 3D control: Cgl3DTeethCtrl – the main class containing all the methods that can be called from other programs. The attributes of the ActiveX class that are shown on the Figure 7 as ActiveX Class and methods, GUI classes and finally, the 3D model rendering classes that will render the 3D model on the screen. The simplified architecture is shown below (see Figure 7) representing an overall design of the ActiveX component.



*FIGURE 7. 3D ActiveX dental charting control's architecture.*

ActiveX class can be considered as the frame, i.e. a drawing canvas which allows other classes and its methods to draw the different kinds of content or user interface elements and 2D controls and 3D graphics as will be demonstrated later. Also, there are 3D model rendering functions that initialize and draw the 3D model on the screen. The functions that draw, initialize the 3D rendering scene and draw user interface controls will be described later in the next subchapters. The next important part of

design implementation is the user interface controls rendering. All controls are drawn using the same API as for the 3D teeth model drawing except that the controls are two-dimensional graphics and the teeth as a 3D model, i.e. a vector-based teeth model and all coordinates can be described in terms of vectors and their directions are associated with those vectors. The XML based GUI is responsible for loading the description of the user interface and drawing the 2D controls based on that description from an XML file that will contain a hierarchical textual map composed as nodes in order to tell to the OpenGL drawing system how to render a position of the controls on the screen. The XML-based user interface will also handle the events that will occur during the execution and life-time of the ActiveX control. All classes were implemented and tested in Microsoft Visual Studio IDE. The most important part is the OpenGL rendering functions that create and draw the model on the screen using the function calls that connect the vertices specified in the object file creating polygons and applying the materials to those polygons to form a human teeth model. The GUI classes create timers, mutexes and as well as the logger capabilities to improve the debugging process. The GUI classes incorporate reading the configuration file capabilities to create a contemporary and sophisticated look of the newly created user interface controls.

From the point of view of the developer, the ActiveX is a container of other functions, public methods and classes, to create a user-defined drawing and other things that are necessary to be displayed on the screen. In this case this technology is essential because it allows integrating different classes to facilitate the appearance of 3D graphics on the screen. The ActiveX control also stores the teeth model coordinates – vertices, vertex indices, texture information, materials information, light sources and so on. The GUI classes are also integrated into the ActiveX container or simply speaking control, but in terms of a software development process, this control can be described as the container for other classes and application programming interfaces. So the ActiveX was implemented for those purposes.

## 4.2 ActiveX control

During the initial planning of the project, it was decided to begin the development process with the ActiveX control implementation and then continue with a 3D model loading. And for the final step - developing and integrating the code of the UI classes into the methods of ActiveX class so that the model manipulation would be possible. So the first phase of the initial planning started with the ActiveX control development. And the implementation process of the ActiveX class is described below. The implemented class is located in source and header files.

From the point of view of the developer, the ActiveX is a container that contains the classes, public methods, functions and variables that are accessible by other programs integrating that container or the control. The main program, which embeds the ActiveX container, will e.g. call the necessary functions or will get the output from that container. In this project the ActiveX component is the main place where all classes and drawing methods reside. For instance, drawing a graphical user interface, loading a 3D-teeth model, calculating geometry information, rendering and displaying the 3D model, parsing the XML configuration file, loading the vertices coordinates and many more – all this will be handled by the ActiveX class which is called `glActiveXCtrl`. This class incorporates different methods that are accessible by the other programs that will embed this control.

First, when the component is loaded into the program the constructor of that class is called to initialize all variables and perform basic operations that will prepare control for the operational state. The constructor is called the `glActiveXCtrl()` and its name is derived from the name of its class. Below there is a code that demonstrates the implementation of this class.

```
// CglBasicCtrl::CglBasicCtrl - Constructor

CglActiveXCtrl::CglActiveXCtrl()
{
    InitializeIIDs(&IID_DglBasic, &IID_DglBasicEvents);
    // TODO: Initialize your control's instance data here.
}
```



After the constructor has been called, the function which has to be called next is a `CglActiveXCtrl::OnCreate()` method called by the container. This function initializes drawing context to execute OpenGL and Windows API drawing functions in order to enable rendering context to execute the OpenGL function calls. Also this function is responsible for choosing the right pixel format available on your system. This pixel format tells you in which mode your video card will operate and what kind of rendering facilities your video card offers. For example, a pixel format tells you whether a double buffering is enabled or not, whether OpenGL is supported on your system or not. It sets the color depth of your screen in a 24- or 32-bit mode. It also enables a stencil buffer or a z-buffer and many other features that can be found on Microsoft's developer site network. The brief description of that class `CglActiveXCtrl` and the method `OnCreate()` of that class is provided in appendix 2 as a UML diagram and the example source code of that class is also provided (see appendix 1). The next function that is called is `wglCreateContext()`, which creates the rendering context for OpenGL drawing commands. After that the function `InitGL()` is called to set the rendering context to make it current. Then it initializes a rotation angle of the model, a scale factor of the 3D teeth model and a screen position of the model in terms of coordinates *x*, *y* and *z*. After a perspective is set, then a line and polygon smoothing is enabled to make a realistic appearance of the teeth model. For the next step light sources and colors are set to make the teeth model highlighted. Then color materials are enabled by calling `glColorMaterial()` to apply material texture to the teeth model. Then a light model is set using a `glLightModeli()` function to highlight the teeth model to enable a realistic and smooth appearance of the 3D scene on the display. After that the matrices and perspective are set by calling `glMatrixMode()` and `glLoadIdentity()` respectively.

When the initial OpenGL rendering scene settings have been set, the method `OnPaint()` is called by the container application to draw the content of the ActiveX control containing the 3D coordinates of the teeth model and that's why the function `OnPaint()` is essential. The `OnPaint()` method is responsible for drawing, rendering, applying material to the teeth model and reassembling the vertices information into one solid teeth model and displaying that model on the screen using OpenGL functions. First of all, the `OnPaint()` method identifies what is the size of its container and where this method needs to draw the 3D teeth model. Then it creates a 3D scene of the required size by using a `GetControlSize()` function and then it returns back the required size of its container window or container program to the `OnPaint()` function. Then the viewport is set to define the size of the

model in pixels and its placement in the 3D scene. Then features such as a zoom factor, a scale and a frustum, also known as a perspective view, are set. And then the drawing loop is executed connecting all the vertices of the 3D teeth that are stored in a three-dimensional array as coordinates of x, y and z vertices in the coordinate space. Connecting all the vertices is done using the function `glBegin(GL_TRIANGLES)` that tells to the OpenGL state machine that it needs to connect the vertices in the form of the triangles to create the final triangle polygons. Basically, the whole teeth model is the interconnection of the triangle polygons that are covered by the color materials and highlighted by the light sources that are set by the OpenGL state machine. The scaling of the model is done using a `glScalef()` function.

### **4.3 3D-Teeth model**

A human 3D-teeth model was created by using a 3DS Max modeling program. First, a 2D picture of the jaws was taken as the foundation for the 3D model. Then step-by-step using the 3DS Max tools the contours of the teeth were drawn following the 2D image of the teeth. Then the model was converted into an editable poly mesh. After that operation every tooth was traced by a Bezier tool that is necessary when tracing the line around the needed object that has to be created in the 3D space. The picture was taken as the example of the teeth. So the aim to make realistic teeth is achieved using the high-resolution 2D image in a jpeg format as the template to create the teeth. 3D Max allows using the Bezier Spline tool to form the 3D model based on a 2D image. So the polygonal model is formed as the interconnection of the lines that are produced by this Spline tool that traces every single point of the 2D image and converts the pixels into the 3D space coordinates that are calculated by the 3DS Max automatically as long as the Spline tool is in use. The imported 2D images of the teeth are shown on the Figure 8 below. This method of creating the 3D object from a two-dimensional image is called a polygonal modeling and it is applicable in cases where the realistic appearance of the 3D object is required like e.g. in the case with human teeth. The polygonal modeling is one of the most commonly used techniques nowadays in the area of computer aided modeling and design. Therefore, this technique was chosen in the implementation phase of the teeth modeling. Steps that were taken in the polygonal modeling of the teeth model can be described as

step-by-step interconnecting the point of the two-dimensional teeth picture between each other and producing results on the screen so that every point in the 2D picture is identical to the 3D model with an additional depth coordinate z in order to make your model look realistic. In order to break the structure of the teeth, the Subdivide tool in the 3DS max was applied to achieve the required results. Material colors were taken from the pictures to achieve the maximal level of the realism. In that way the polygonal teeth mesh were implemented in the 3DS Max modeling program.

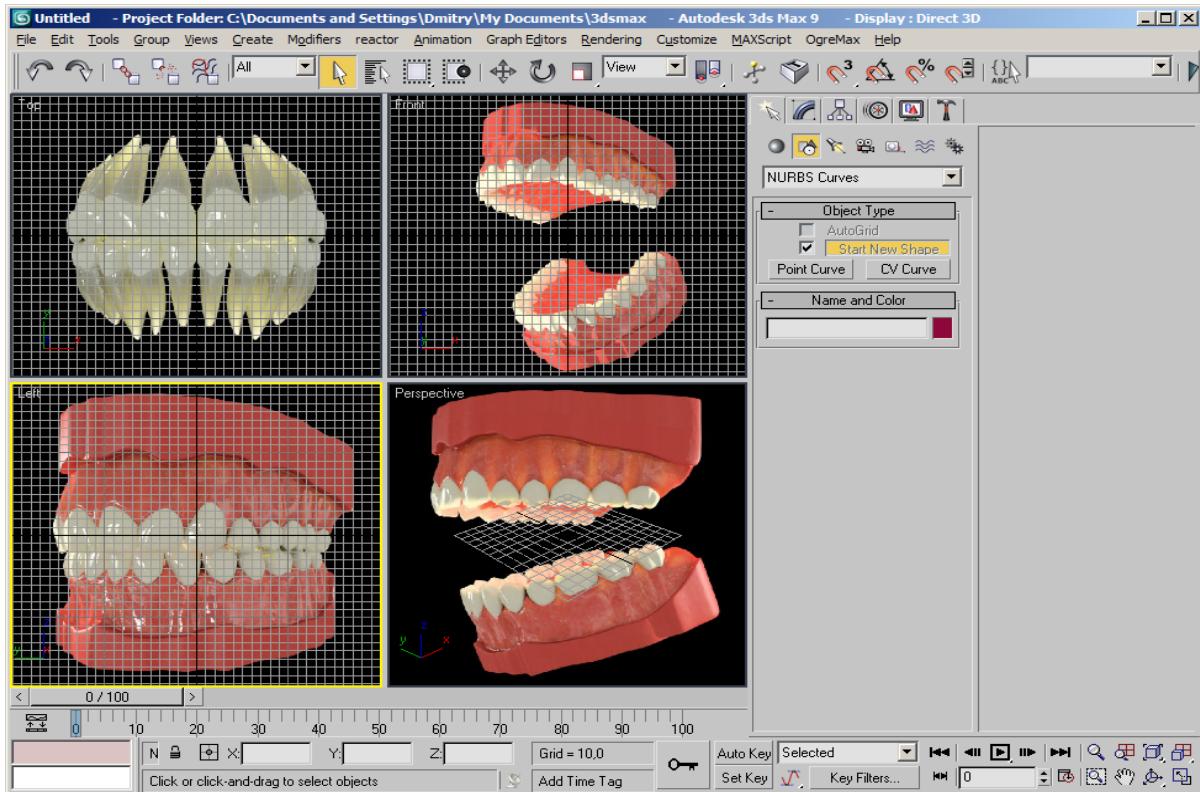


FIGURE 8. Imported 2D images of the human teeth in 3DS Max.

#### 4.4 XML based user interface

In order to shorten the development time, the graphical user interface implementation was taken from the Nvidia Nature Scene demo scene (Nvidia 2005, date of retrieval 3.04.2012) to demonstrate the usage of the shaders technology to produce a realistic scene of the nature. This GUI implementation was borrowed with some modifications in order to integrate the existing project into the implementation of the ActiveX component.

The GUI implementation consists of the classes that implement the standard controls like buttons, text edits, sliders, checkboxes, separators, radio buttons and so on. All user interface controls are drawn using the OpenGL API. That was the main reason why this GUI implementation was chosen. In order to preserve the compatibility between the rendering 3D teeth model and drawing the user interface controls OpenGL API were used throughout the implementation phase.

The main class in the GUI implementation is the GUIRectangle base class for all the controls that were developed. As initially all controls are in some way rectangles with the drawn content inside of them, then the rest of all UI classes are inherited from the base class. The following classes GUIButton, GUISlider, GUITextEdit are inherited from that base class GUIRectangle. For the next step all these controls are drawn by OpenGL API based on the XML configuration file that describes the layout, color, position and the appearance of the user interface on the screen. This file tells to the OpenGL state machine how the controls should be drawn and placed on the screen. This file transfers the coordinates and color information of the controls to the OpenGL machine through the XML parsing library that reads the XML configuration file and sets all the variables that are accessible by the OpenGL drawing functions in order to produce the rendering of the controls on the screen. The controls can be seen in the appendix 3. This figure demonstrates the controls that are produced by the OpenGL API based on the parsed information from the XML configuration files. The source code of the GUI interface can be found in appendix 4. The example XML configuration file can be found in the appendix 5.

## **4.5 UI Screen**

The user interface screen is divided into two parts: the first one, where the rendering of the teeth model happens and the second one, where the UI controls reside. The reason behind this is to prevent the detection collision between the 3D model events and user interface events. The user interface event handler needs to know in what coordinate space the click happens in order to properly respond to that event.

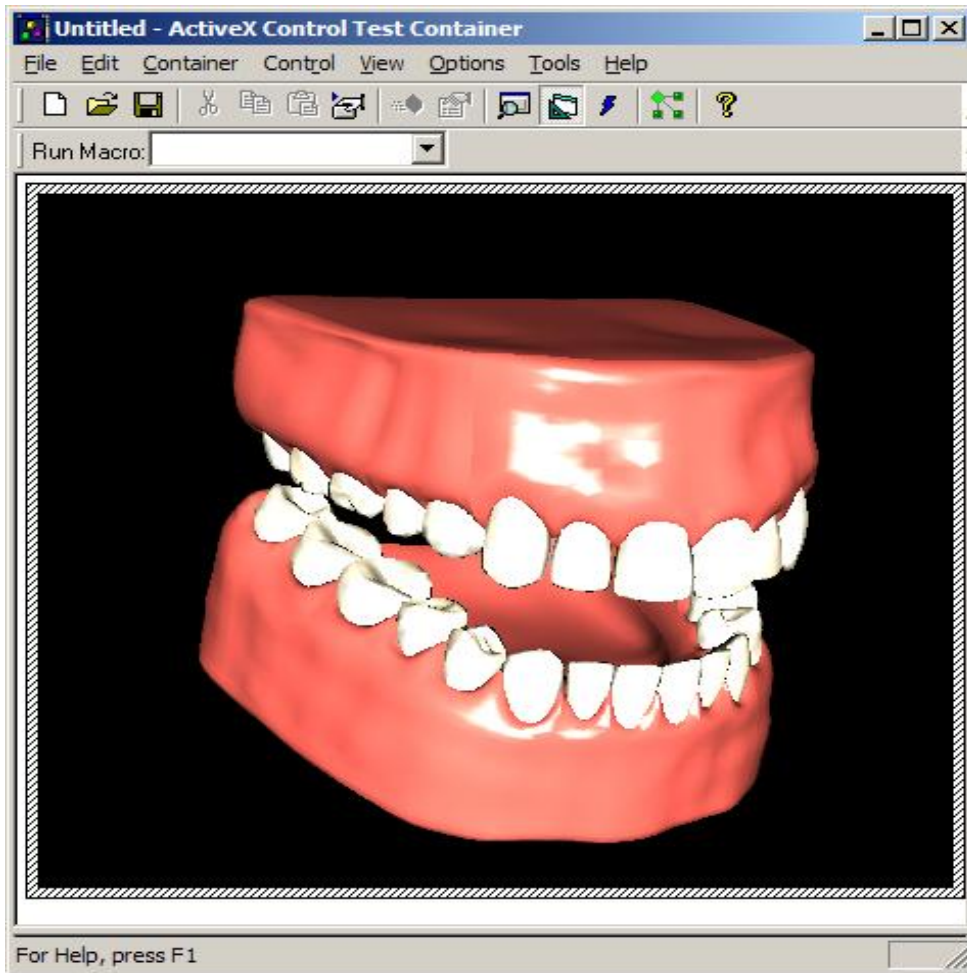
The 3D teeth model is rendered in its own coordinate space, following proper design guidelines that allow the proper handling of the events associated with the teeth model. For example, if the user clicks on the model and then holds the left or right mouse button and tries to rotate the model, so in that way there is no collision between the user control events and teeth model events. OpenGL also creates the rendering contexts for us in order to overcome all mistakes that are associated with the handling of the user interface events and manipulating the 3D model at the same time.

### **4.5.1 Teeth model rendering**

The three-dimensional look is implemented via drawing the 3D-teeth model on the screen using the OpenGL API functions that directly render and apply color material to the target 3D model. Also, the multiple light sources and perspective are set to produce the feel of the depth to the user.

From the point of view of the programmer, the teeth model rendering was implemented using a loop that continuously interconnects the vertices into the polygons using the OpenGL commands `glBegin()` and `glEnd()` to begin and end rendering respectively into the framebuffer, and then a video card sends the content of the framebuffer onto the display. The basic shape of each polygon in the teeth model is a triangle that is formed from 3 coordinates – x, y and z. Those coordinates are stored as the data of floating-point type in three dimensional arrays. There are approximately two hundred thousands of triangles in the human teeth model designed in 3DS Max. But considering the processing power of modern video cards and processors that is not an issue at all. The loading of the coordinates happens when the rendering loop begins and then the coordinates are transferred into

the video card to produce the 3D teeth model on the screen. All that is done with the help of OpenGL API and the driver of the video card on the target system. In the implementation process the method of rendering the 3D model was called DrawScene(). The final source code of the DrawScene() method can be found in appendix 6. 3D teeth model inside of the ActiveX control is shown on the Figure 9 below.



*FIGURE 9. 3D human teeth model.*

#### **4.5.2 Teeth model manipulation**

The model manipulation of the teeth model and eventually the user interface controls handling is accomplished using a mouse. There is one method that is responsible for mouse events handling. This method is called `OnMouseMove()` and the main purpose of this function is to change the rotation angle of the model and redraw the scene when the angle is changed by the user. The value of an angle variable is changed every time when the user is moving the mouse pressing down left-button at the same time. The source code of this method can be found in the appendix 1.

#### **4.5.3 XML UI settings**

The settings that are associated with the user interface of the ActiveX control are stored in the XML file. This file contains the description of the user interface, which has to be displayed by the OpenGL API. There are the nodes that tell to the control manager how the controls should be displayed on the screen and in which order, the position in terms of the screen coordinates, the control appearance, and so on. The XML file also tells to the layout manager how the control should be placed on the screen relatively to the other controls. All that is done through the layout manager implemented in a C++ programming language. The example of the XML configuration file can be found in the appendix 5.

## 5. TESTING

The application was tested using the ActiveX Control Test Container tool available on every Windows workstation. During the testing phase, I tested the performance and robustness of the ActiveX control. Also, the compatibility with different operating systems was tested due to the fact that different Windows OS versions use different drivers to render the 3D graphics on the screen. The main point of testing was to see whether the performance of the final version is sufficient to target that ActiveX control to the target platform and whether the application will be compatible with the different video cards and Windows operating system version. And the result of the testing on different platform was successful due to the fact that in the implementation process the standard features of the modern video cards and OpenGL API were used to enable the compatibility between different video cards and the OpenGL version supported by video cards' drivers.

The testing has revealed that the ActiveX control functions properly without any delays or crashes. Overall the performance of the final ActiveX component is high. The reason for that is that during the teeth model design phase, the teeth model was designed in such a way that it could easily be rendered even by the obsolete and outdated software. In order to check the performance of the rendering, the Frames per Second counter class was taken from the Nvidia Nature Scene demonstration kit. The frames per second performance testing class measures the amount of frames that are produced in one second so in that way an overall performance of the final application can be tested. When this class was integrated into the rendering process of the ActiveX control, it revealed that the rendering speed is acceptable on the most systems and the speed ranges from 60 up to 80 frames per second on different platforms depending on the hardware, i.e. the performance of the video card and processor on that platform.

Methods were called using the ActiveX Control Test Container tool. The main method that was the most important is a DrawScene() method. That method is responsible for the rendering and displaying the GUI and 3D teeth model on the screen. Also, the mouse manipulation of the teeth model by the user was tested successfully. During the testing phase DrawScene() was working without any problems and that concluded the testing phase.



## **6. FUTURE DEVELOPMENT**

At this moment the application lacks some functionality, for example changing the color of the specific region in a tooth and a more flexible manipulation of the two parts of the human jaw, upper and lower jaw. There is no animation of the teeth model and there is e.g. no possibility to change the rotation angle of the upper and lower jaw, just one model as whole can be rotated. But all this can be changed in a future development phase. New features can be added, too. All this will benefit the user's experience with the application and will improve the overall usability of the interface.

There are some possible future improvements that can be developed and added to the existing system in the near future. Although there are some features that are very hard to integrate into the existing system, it is worth mentioning them, too.

### **6.1 Functionality**

The existing functionality of the developed ActiveX control can be enhanced by adding such features as rotating the upper and lower jaw of the teeth model in different directions, different views, changing the color of the teeth materials, changing the color of an individually selected tooth, animation and many more. Also, the existing 2D teeth chart can be combined with a newly developed 3D teeth model, which can be used simultaneously with the 2D-teeth chart.

There are features that will extend the functionality even more, for example viewing the X-Ray images of the human skull and also the teeth analysis chart of the internal structure which enables the dental professional to see the state of the patient's health in real-time. That will help to plan the treatment process beforehand, in order to predict the possible issues with the patient's teeth in the near future. Overall, this will improve the treatment process of a dental professional. There are also other possibilities to improve the existing 3D dental charting. All of them are described below.

## 6.2 DICOM Viewer

The DICOM images of the human teeth can also be viewed by an integrating DICOM viewer added into the existing ActiveX control in order to analyze the patient's state of health. DICOM can be described as a global medical standard that is widely used in all hospitals worldwide. Developed in 1993 it is used to ensure the compatibility and interoperability between medical systems, which are used to produce, display, store, send and print medical images (National Electrical Manufacturers Association 2011, date of retrieval 5.04.2012). Dental professionals and physicians at the hospitals and healthcare centers can benefit from DICOM by having a better access to medical images and treatment reports so that will allow them to make better and faster diagnosis. And on the other hand, patients can obtain the medical results wherever they are. Medical diagnosis results can also be sent to other healthcare centers following the DICOM standard. The example of the DICOM Viewer that can be integrated into the existing ActiveX control is shown on Figure 10.

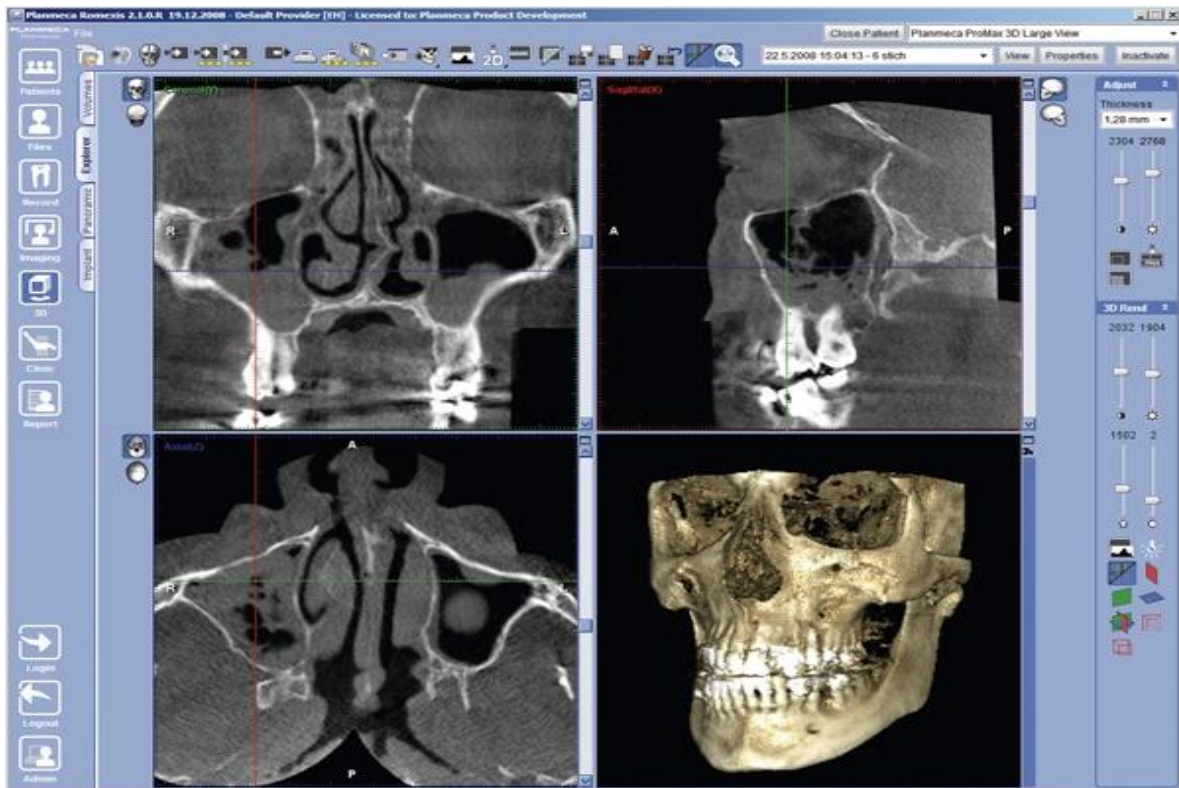


FIGURE 10. DICOM Viewer used in Planmeca ProMax 3D computed tomography software.

### **6.3 UI Settings**

Currently, UI settings are placed in the XML configuration file. And the content of this configuration file can be changed only before the loading of the ActiveX control in the container program. So the improved version can be extended by adding a dynamical user interface design, or the user can choose different skins and layouts from the user interface. Also, for example, the free movement of the user interface controls would be possible.

Another possibility is to add a voice recognition feature into the existing ActiveX controls, so the user can manipulate the teeth model by voice commands and enable this voice manipulation system by the user interface controls. This feature, for instance, can be implemented using Open Audio Library.

### **6.4 Ray-tracing rendering**

The ray-tracing rendering method is used when there is a necessity to produce realistic 3D graphics of high degree. This technique is capable of producing very realistic objects on the computer screen but very high computational costs are needed to render such kind of graphics. One of the most popular ray-tracing is the Monte-Carlo ray tracing algorithm.

Realistic image synthesis is increasingly important in areas such as entertainment (movies, special effects and games), design, architecture and more. A common trend in all these areas is to request more realistic images of increasingly complex models. Monte Carlo ray tracing based techniques are the only methods that can handle this complexity. Recent advances in algorithms and compute power has made Monte Carlo ray tracing the natural choice for most problems. This is a significant change from just a few years back when the (finite element) radiosity method was the preferred algorithm for most graphics researchers. (SIGGRAPH 2003, date of retrieval 6.04.2012)

## 6.5 Teeth charting

The existing teeth charting can be improved by integrating the existing two-dimensional teeth charting that were used for many years in the WinHIT software which is developed by In Net Oy company. The next-generation ActiveX component can be developed in such a manner that it would be able to use a 2D/3D and DICOM viewer at the same time by implementing a multi-tab user interface that would be suitable for usage in the WinHIT software. The example can be seen on Figure 11.

A newly developed interface would bring to the users an increased level of usability by using the 2D/3D and DICOM viewer at the same time. So there is a need for any external software for accomplishing the task, like browsing patient's history and searching the scan images of patient's teeth. With a new interface, the user would be able to do it just in a single window.

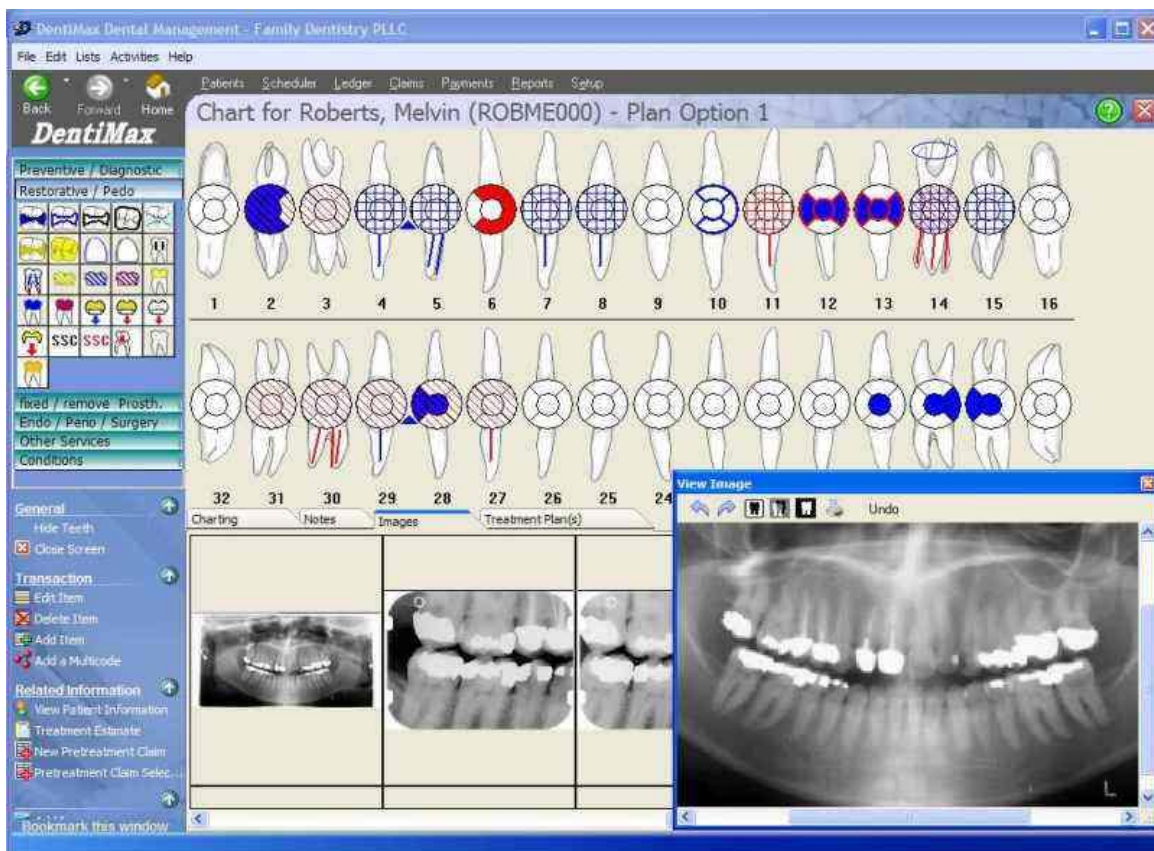


FIGURE 11. User interface of DentiMax dental management practice software.

## 7. CONCLUSION AND RESULTS

The developed 3D ActiveX dental charting demonstrates a new way of presenting the results of a dental treatment to the dental professionals and physicians who are working in the area of dental medicine and orthodontology. It is very flexible and portable so it can be deployed and run almost on any Windows workstation with a contemporary video card, thanks to the OpenGL API, which is extremely portable and capable of producing a realistic teeth model on the computer display without any demands from the video subsystem of the target device.

During the development phase of the project I learned how to properly design a software that will be used by dental professionals in the area of medicine and healthcare. All this gave me a deep understanding of how the software requirements of the medical program components should be met and what kind of programming languages, software development tools, specifications and technologies I had to use. Additionally, I learned the proper project development planning and overall my development and coding skills increased, too. Despite some issues that occurred during the implementation phase of the project, the iterative and incremental methodology gave me a better understanding of the proper project development process.

Personally, I feel that the whole process of development was demanding and challenging but my understanding of the software development for medical area has grown incredibly. This was achieved through thorough discovering of the technologies and specifications that were used in the implementation phase of the project. I became more experienced in using the iterative and incremental development approach and this approach indeed also showed my weaknesses and strengths that I need to be aware of.

Currently, the ActiveX component lacks many features, such as highlighting a decayed or problematic region of the teeth that requires treatment, but it will be finished in the future development phase. But the current version is a solid foundation for the future versions of this project and I hope that there are many ways of improving the existing ActiveX component. Nevertheless, the aim of the project was achieved successfully. I hope this project will also bring benefits to the company in the future.

## 8. DISCUSSION

After completing the project, the main aim was achieved successfully. At the beginning of the development phase the specifications of the project were unclear but when I progressed in my development phase I began to realize the issues that had to be solved. For instance, during the ActiveX implementation it was hard to combine a 3D-teeth model and GUI controls that were rendered in the same rendering context. When I realized that it was a wrong approach, I decided to implement different rendering contexts, one for the user interface controls rendering and another one for the teeth model rendering. The testing revealed that I was on the right way of thinking. This approach helped me to overcome issues that would occur if I would implement just one rendering context for OpenGL drawing commands. Considering my previous software engineering experience, I realized that I need to improve my software designing skills constantly due to the fact that in every project there is supposed to be an approach that is suitable for it. And that is very difficult to do.

In my personal opinion this project will increase the level of usability of the user interface for the dental professionals, who will use this ActiveX component in their dental practices, by improving the appearance of the dental treatment procedures at healthcare centers and dental clinics all over Finland. The newly developed ActiveX dental charting can be integrated in many software products developed by In Net Oy company. This was achieved by using cross-platform and portable technologies like ActiveX and OpenGL.

In the near future this project will be enhanced by adding additional functionality that was described previously in the preceding chapters. For instance, the DICOM viewing feature will definitely benefit the dental professionals who want to use just one program that accomplishes many tasks in one window.

Personally, the development of a such component broadened my horizons and increased my level of competence in the area of 3D modeling. Now I can proudly state that I am capable of doing such things like 3D modeling, and writing programs that will be used by dental professionals all over the Finnish healthcare system.

## 9. REFERENCES

1. American Dental Association 2012. Permanent Tooth Eruption Chart.  
[http://www.ada.org/sections/publicResources/pdfs/chart\\_eruption\\_perm.pdf](http://www.ada.org/sections/publicResources/pdfs/chart_eruption_perm.pdf)  
Date of retrieval: 5.02.2012
2. Autodesk 2012. Autodesk 3ds Max Products.  
<http://usa.autodesk.com/3ds-max/features/>  
Date of retrieval: 6.02.2012
3. Autodesk 2007. 3DS Max User's Reference.  
[http://download.autodesk.com/us/3dsmax/learning\\_path/3dsmaxref\\_vol1.pdf](http://download.autodesk.com/us/3dsmax/learning_path/3dsmaxref_vol1.pdf)  
Date of retrieval: 6.02.2012
4. Blinkhorn A., Choi C. & Paget H. 1998. An investigation into the use of the FDI tooth notation system by dental schools in the UK. *Eur J Dent Educ* 2. Page(s): 39–41
5. D. Chappell. *Understanding ActiveX and OLE*. 1996.  
Microsoft Press. U.S. ISBN-10: 1572312165
6. H. Chen & Z.. Zheng. 2008. ISC SCT '08. International Symposium.  
Issue Date: 20-22 December 2008. Volume: 2. On page(s): 380 - 382  
Print ISBN: 978-1-4244-3746-7
7. Cormen, C. Leiserson, R. Rivest & C. Stein. 2009. *Introduction to Algorithms*. MIT Press.  
3<sup>rd</sup> Revised edition. ISBN-10: 0262033844
8. A. Cockburn 2008. *Using both incremental and iterative development*.  
<http://alistair.cockburn.us/Using+both+incremental+and+iterative+development>  
Date of retrieval: 19.03.2012

9. FDI World Dental Federation 2012. FDI Two Digit-Notation.  
<http://www.fdiworldental.org/two-digit-notation>  
Date of retrieval: 5.02.2012
  
10. A. R. Hussain, K. Kachymalay & A. A. Aziz. 2010. Information Sciences Signal Processing and their Applications (ISSPA). 10th International Conference on Signal Processing.  
Digital Object Identifier: 10.1109/ISSPA.2010.5605509. Page(s): 794 - 797
  
11. In Net Oy 2012. Hammashoidon Integroitu Tietojärjestelmä.  
[http://www.winhit.fi/downloads/WinHit\\_esite1211-6.pdf](http://www.winhit.fi/downloads/WinHit_esite1211-6.pdf)  
Date of retrieval: 3.02.2012
  
12. A. Jones & W. Sherrod 2011. Beginning DirectX 11 Game Programming. Delmar Cengage Learning. 1st edition. ISBN-10: 1435458958
  
13. M.F. Marotta, P. Phanichphant, P. Malatack, T. Shah, G. Price, T. Thyvalikakath, T. Schleyer & J. Hong. 2007. Design and evaluation of 3d models for electronic dental records. Conference on Human Factors in Computing Systems. Page(s): 2567-2572. San Jose, USA.
  
14. Microsoft 2012. ActiveX Control Test Container.  
[http://msdn.microsoft.com/en-us/library/ms241446\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/ms241446(v=vs.80).aspx)  
Date of retrieval: 7.02.2012
  
15. Microsoft 2012. Introduction to ActiveX Controls.  
[http://msdn.microsoft.com/en-us/library/aa751972\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa751972(v=vs.85).aspx)  
Date of retrieval: 3.02.2012
  
16. H. Min & X. Xiaoyan. 2007. Software Development for Electromagnetic Tomography Image Reconstruction Using ActiveX. Shanghai Maritime University, Shanghai. On page(s): 2-559 - 2-562. Print ISBN: 978-1-4244-1136-8. Digital Object Identifier: 10.1109/ICEMI.2007.4350742.



17. Microsoft 2006. MFCOpenGL a Generic MFC OpenGL Code sample.  
<http://support.microsoft.com/kb/127071>  
Date of retrieval: 12.03.2012
  
18. Microsoft 2012. MFC ActiveX Controls: Painting an ActiveX Control.  
[http://msdn.microsoft.com/en-us/library/552z1tz3\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/552z1tz3(v=VS.90).aspx)  
Date of retrieval: 12.02.2012
  
19. National Electrical Manufacturers Association 2011. Digital Imaging and Communications in Medicine (DICOM) Part 1: Introduction and Overview.  
[http://medical.nema.org/Dicom/2011/11\\_01pu.doc](http://medical.nema.org/Dicom/2011/11_01pu.doc)  
Date of retrieval: 5.04.2011
  
20. Nvidia 2005. Nature Scene.  
<http://developer.nvidia.com/system/files/akamai/gamedev/docs/Nature.pdf?download=1>  
Date of retrieval: 3.03.2012
  
21. W.H. Gruber, M.M. Pollack & D.S. Rosenthal. 1994. Health Care Technology Policy I: The Role of Technology in the Cost of Health Care. Page(s): 166 – 171. Print ISBN: 0-8194-1631-2. Digital Object Identifier :10.1109/HCTP.1994.721291
  
22. Right Hemisphere 2010. Deep Exploration 6.1 CAD Edition Product Brief.  
[http://www.righthemisphere.com/\\_base/static/img/products/callouts/Deep-Exploration-6.1-CAD-Edition-Product-Brief-v01.pdf](http://www.righthemisphere.com/_base/static/img/products/callouts/Deep-Exploration-6.1-CAD-Edition-Product-Brief-v01.pdf)  
Date of retrieval: 7.02.2012
  
23. M. Segal & K. Akeley. 2006. The OpenGL Graphics System: A Specification, version 2.1 edition, OpenGL Architecture Review Board.  
<http://www.opengl.org/documentation/specs/version2.1/glspec21.pdf>  
Date of retrieval: 8.02.2012

24. SIGGRAPH 2003. Monte Carlo Ray Tracing.  
<http://geometry.caltech.edu/~keenan/mcrt-sg03c.pdf>  
Date of retrieval: 6.04.2012
  
25. C. Sinthanayothin & W. Tharanont. 2008. Computerized Algorithm for 3D Teeth Segmentation. International Conference on Electronics and Information Engineering (ICEIE 2010). Pathumthani Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology. Volume: 1. Page(s): 81 – 84. Print ISBN: 978-1-4244-2101-5.
  
26. The Khronos Group 2012. OpenGL API Documentation Overview.  
<http://www.opengl.org/documentation/>  
Date of retrieval: 5.02.2012
  
27. The Khronos Group 2012. OpenGL Platform & OS Implementations.  
<http://www.opengl.org/documentation/implementations/>  
Date of retrieval: 5.02.2012
  
28. The Kronos Group 2012. The OpenGL Graphics System.  
<http://www.opengl.org/registry/doc/glspec42.core.20120119.pdf>  
Date of retrieval: 6.02.2012
  
29. D. Wang, Y. Zhang, J. Hou, Y. Wang, P. Lv, Y. Chen & H. Zhao. 2011.  
iDental: A Haptic-Based Dental Simulator and its Preliminary User Evaluation.  
Beihang University, Beijing. Volume: PP. Issue: 99. Page(s): 1-1. IEEE Transactions on Haptics.  
Digital Object Identifier: 10.1109/TOH.2011.59
  
30. W3C 2003. Extensible Markup Language.  
<http://www.w3.org/XML/>  
Date of retrieval: 14.03.2012

## 10. APPENDICES

APPENDIX 1	THE SOURCE CODE OF CGLACTIVEXCTRL CLASS
APPENDIX 2	THE UML DIAGRAM OF CGLACTIVEXCTRL CLASS
APPENDIX 3	THE XML BASED GUI FIGURE
APPENDIX 4	THE XML BASED GUI SOURCE CODE
APPENDIX 5	XML BASED GUI CONFIGURATION FILE
APPENDIX 6	3D MODEL RENDERING METHOD DRAWSCENE()

## APPENDIX 1

### THE SOURCE CODE OF CGLACTIVEXCTRL CLASS

```
CglActiveXCtrl::CglActiveXCtrl()
{
    InitializeIIDs(&IID_DglBasic, &IID_DglBasicEvents);
    // TODO: Initialize your control's instance data here.
}

void CglActiveXCtrl::OnDraw(
    CDC* pdc, const CRect& rcBounds, const CRect& rcInvalid)
{
    if (!m_pDC)
        return;

COleControl::ValidateRect(rcBounds);

    if (!IsOptimizedDraw())
    {
        // The container does not support optimized drawing.

        // TODO: if you selected any GDI objects into the device context *pdc,
        // restore the previously-selected objects here.
    }

    DrawScene( m_pDC->m_hDC );

}

void CglActiveXCtrl::OnPaint()
{
    CPaintDC dc(this); // device context for painting
    // TODO: Add your message handler code here
    // Do not call COleControl::OnPaint() for painting messages

    DrawScene( m_pDC->m_hDC );
}
```

```

// CglActiveXCtrl message handlers
int CglActiveXCtrl::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    //////////// create control structure
    if (COleControl::OnCreate(lpCreateStruct) == -1)
        return -1;

    lpCreateStruct->style |= ( WS_OVERLAPPEDWINDOW |   WS_CLIPCHILDREN |
    WS_CLIPSIBLINGS |
    CS_OWNDLC | CS_HREDRAW | CS_VREDRAW );

    PIXELFORMATDESCRIPTOR pfd = {
        sizeof(PIXELFORMATDESCRIPTOR),
        1,
        PFD_DRAW_TO_WINDOW |
        PFD_SUPPORT_OPENGL |
        PFD_SWAP_COPY |
        PFD_SWAP_EXCHANGE |
        PFD_DOUBLEBUFFER,
        PFD_TYPE_RGBA,
        24,           // color depth   (RGB)
        0,0,0,0,0,0,
        8,0,0,0,0,0,
        24,           // 24 z-buffer - recommended
        8,0,           // stencil bits -> using stencil
        PFD_MAIN_PLANE,
        0,0,0,0
    };

    if ( !( m_pDC = GetDC() ) )
    {
        MessageBox( "GetDC() failed!", "Error", MB_ICONWARNING );
        return FALSE;
    }

    int nPixelFormat = ChoosePixelFormat( m_pDC->m_hDC, &pfd );

    if ( !nPixelFormat )
    {
        MessageBox( "ChoosePixelFormat failed", "Error", MB_ICONWARNING );
        return FALSE;
    }
}

```

```

if ( !SetPixelFormat( m_pDC->m_hDC, nPixelFormat, &pfid ) )
    {
        MessageBox( "Can't set the pixel format!", "Error", MB_ICONWARNING );
        return FALSE;
    }

m_hRC = wglCreateContext( m_pDC->m_hDC );

if ( !(wglMakeCurrent( m_pDC->m_hDC, m_hRC )) )
    {
        MessageBox( "Make context current failed", "Error", MB_ICONWARNING );

        return FALSE;
    }

InitGL();

SwapBuffers( m_pDC->m_hDC );
wglMakeCurrent( NULL, NULL );

return 0;
}

```

```

void CglActiveXCtrl::InitGL()
{
    xAngle = 0.0f;
    yAngle = 170.0f;
    zAngle = 50.0f;

    xScale = 2.6f;
    yScale = 2.6f;
    zScale = 2.6f;

    this->zoomFactor = 1.0f;

    this->m_bLButtonPressed = FALSE;
    this->m_bLButtonUp = FALSE;
    this->m_bAnimation = FALSE;

    this->xTranslatePos = 0.0f;
    this->yTranslatePos = 0.0f;
    this->zTranslatePos = -6.4f;
}

```

```

glFrontFace( GL_CCW );
glEnable(GL_CULL_FACE);
glHint( GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST );

glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
glShadeModel( GL_SMOOTH );

glEnable(GL_POINT_SMOOTH );
glEnable( GL_LINE_SMOOTH );
glEnable(GL_POLYGON_SMOOTH );
glAlphaFunc( GL_GREATER, 0.1f );
glEnable( GL_ALPHA_TEST );

glHint( GL_NORMALIZE, GL_NICEST );
glHint( GL_COLOR_MATERIAL, GL_NICEST );

GLfloat glfLightAmbient[] = {0.1f, 0.1f, 0.1f, 1.0f};
GLfloat glfLightDiffuse[] = {1.2f, 1.2f, 1.2f, 1.0f};
GLfloat glfLightSpecular[] = {0.9f, 0.9f, 0.9f, 1.0f};

glLightModeli( GL_LIGHT_MODEL_COLOR_CONTROL, GL_SEPARATE_SPECULAR_COLOR );
glLightfv(GL_LIGHT0, GL_AMBIENT, glfLightAmbient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, glfLightDiffuse);
glLightfv(GL_LIGHT2, GL_SPECULAR, glfLightSpecular);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
glEnable(GL_LIGHT2);
glHint( GL_LIGHTING, GL_NICEST );

glColorMaterial( GL_FRONT, GL_AMBIENT_AND_DIFFUSE );

glClearDepth(1.0f);
glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
glHint( GL_DEPTH_TEST, GL_NICEST );
glHint( GL_LEQUAL, GL_NICEST );
glLightModeli(GL_LIGHT_MODEL_AMBIENT , 2);

glMatrixMode( GL_PROJECTION );
glPushMatrix();
glLoadIdentity();
}

```

```
void CglActiveXCtrl::OnMouseMove(UINT nFlags, CPoint point)
{
    if ( this->m_bLButtonPressed )
    {
        int posX = (int) point.x;
        int posY = (int) point.y;

        this->xAngle -= ( (GLfloat) point.x * 5.0f) / 1200.0f;
        this->yAngle += ( (GLfloat) point.y * 5.0f) / 1200.0f;
        this->zAngle += 5.0f;

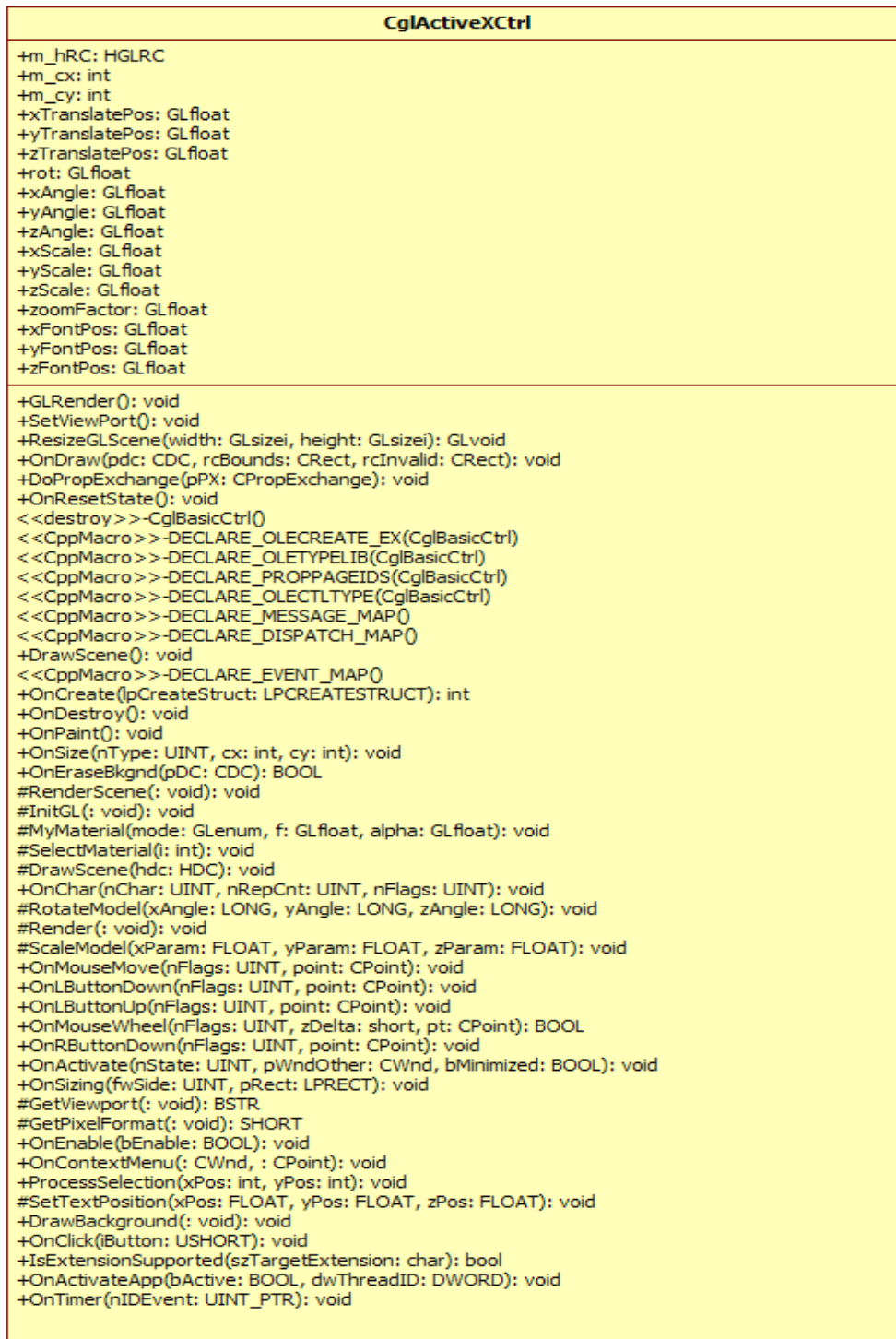
        DrawScene( m_pDC->m_hDC );
    }

    COleControl::OnMouseMove(nFlags, point);
}
```



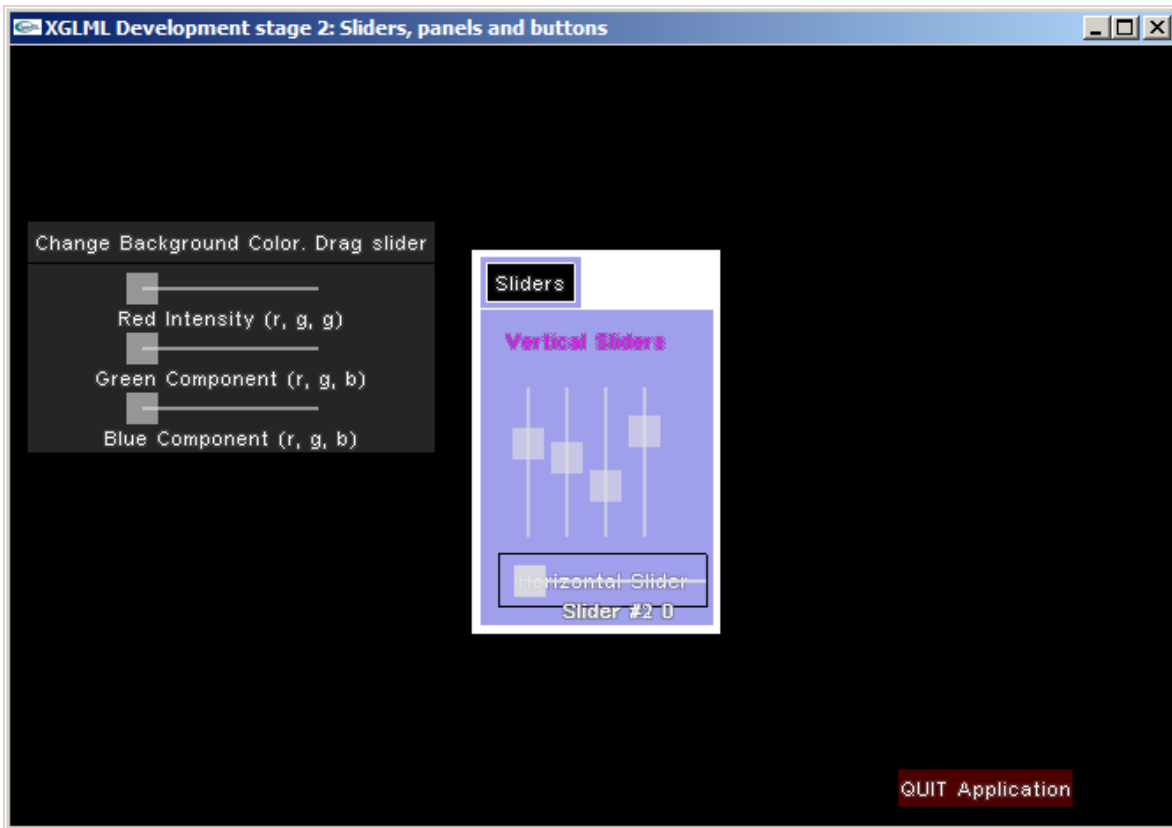
## APPENDIX 2

### THE UML DIAGRAM OF THE CGLACTIVEX CLASS



## APPENDIX 3

### XML BASED GUI FIGURE



## APPENDIX 4

### THE XML BASED GUI SOURCE CODE

THE SOURCE CODE IS TAKEN FROM NVIDIA NATURE SCENE DEMO

<http://developer.nvidia.com/content/nature-scene-code-sample>

#### SAMPLE GUIRectangle class

```
GUIRectangle::GUIRectangle(const char*callback)
{
    setCallbackString(callback);
    setDimensions(5, 5);
    setPosition(0.5f, 0.5f);

    lastAction = 0;
    widgetType =UNKNOWN;
    anchor    = CENTER;
    parent    = NULL;
    z        = 1;

    mouseOver = false;
    released  = false;
    pressed   = false;
    clicked   = false;
    focused   = false;
    visible   = true;
    active    = true;
    update    = false;
}

void GUIRectangle::setParent(GUIRectangle *parentArg)
{
    parent = parentArg;
    update = (parent != NULL);
}

GUIRectangle *GUIRectangle::getParent()
{
    return parent;
}
```

```

}

int GUIRectangle::getWidgetType()
{
    return widgetType;
}

void GUIRectangle::forceUpdate(bool updateArg)
{
    update = updateArg;
}

bool GUIRectangle::loadXMLSettings(const TiXmlElement *element)
{
    if(!element)
        return false;

    const char *cbString = element->Attribute("callbackString");
    cbString = cbString ? cbString : element->Attribute("name");

    if(cbString)
        setCallbackString(cbString);
    else
        return Logger::writeErrorLog(String("Need a <name> or <callbackString> attribute, check ")
            + element->Value());

    setAnchorPoint(element->Attribute("anchorPoint"));
    setVisible(XMLArbiter::analyzeBooleanAttr(element, "visible", true));
    setActive(XMLArbiter::analyzeBooleanAttr(element, "active", true));

    for(const TiXmlElement *child = element->FirstChildElement();
        child;
        child = child->NextSiblingElement() )
    {
        const char * value = child->Value();

        if(value)
        {
            if(!strcmp(value, "Position"))
                setPosition(XMLArbiter::fillComponents2f(child, position));

            if(!strcmp(value, "Dimensions"))
                setDimensions(XMLArbiter::fillComponents2f(child, dimensions));
        }
    }
    return true;
}

```

## APPENDIX 5

### XML BASED GUI CONFIGURATION FILE

#### THE XML CONFIG FILE IS TAKEN FROM NVIDIA NATURE SCENE DEMO

<http://developer.nvidia.com/content/nature-scene-code-sample>

```
<Panel visible = "true"
  name = "Main Frame">

  <Texture type = "TEXTURE_2D"
    path = "GUIElements.PNG"
    mode = "MODULATE"
    mipmap = "true">

    <Wrap s = "REPEAT"
      t = "REPEAT" />

    <Filter mag = "LINEAR"
      min = "LINEAR_MIPMAP_LINEAR" />
  </Texture>

  <Font path = "ArialSmall.tga" />
  <Panel description = "ColorSliders.xml" />

  <Panel name = "Parent"
    layout = "XAXIS"
    drawBounds = "false"
    drawBackground = "true">

    <BordersColor x = "255" y = "255" z = "255" />
    <BGColor x = "255" y = "255" z = "1.0" w = "1.0"/>
    <Texture type = "TEXTURE_2D" path = "opengl.jpg" />
    <TextureRectangle x = "0.0" y = "0.0" z = "1.0" w = "1.0" />

    <TabbedPanel callbackString = "tabs">
      <TabButtonsBordersColor x = "255" g = "255" b = "250" />
      <TabButtonsColor x = "0" g = "0" b = "0" />
      <BordersColor x = "255" y = "255" z = "255" />
      <BGColor x = "127" y = "127" z = "230" />
    </TabbedPanel>
  </Panel>
</Panel>
```

```

    <Panel description = "Sliders.xml" />
    <Panel description = "Buttons.xml" />
    <Panel description = "Editable.xml" />
    <Panel description = "Checkable.xml" />
    <Panel description = "Others.xml" />
  </TabbedPanel>
</Panel>
<Panel name      = "EParent"
  anchorPoint = "CENTER">
  <BordersColor x = "255" y = "255" z = "255" />
  <BGColor      x = "50" y = "50" z = "50" />
  <Position     x = "-10" y = "-10"/>

  <Button callbackString = "exit">
    <EdgeOffsets x = "1.0" y = "1.0" />
    <Text  string = "QUIT Application" />
  </Button>
</Panel>
</Panel>
<TexCoordsDesc xStart = "201" yStart = "2" xEnd = "233" yEnd = "34" type = "SLIDER"/>
<TexCoordsDesc xStart = "167" yStart = "2" xEnd = "199" yEnd = "34" type = "CHECK_BOX"/>
<TexCoordsDesc xStart = "131" yStart = "1" xEnd = "165" yEnd = "35" type =
"CHECK_BOX_MARK"/>
<TexCoordsDesc xStart = "1" yStart = "1" xEnd = "65" yEnd = "66" type = "RADIO_BUTTON"/>
<TexCoordsDesc xStart = "66" yStart = "2" xEnd = "130" yEnd = "66" type =
"CHECK_RB_MARK"/>
</Panel>

```

## APPENDIX 6

### 3D MODEL RENDERING METHOD DRAWSCENE()

```
void CglActiveXCtrl::DrawScene(HDC hdc)
{
    wglMakeCurrent( hdc, m_hRC );

    GLfloat fAspect = (GLfloat) m_cx / (GLfloat) m_cy;

    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();
    gluPerspective(30.0 * this->zoomFactor, fAspect, 1.0, 10.0);
    glViewport(0, 0, m_cx, m_cy);

    glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glTranslatef( xTranslatePos, yTranslatePos, zTranslatePos );

    glRotatef( xAngle, 1.0f, 0.0f, 0.0f);

    glRotatef( yAngle, 0.0f, 1.0f, 0.0f);
    glRotatef( zAngle, 1.0f, 0.0f, 0.0f);

    glScalef( xScale, yScale, zScale );

    int mcount = 0;
    int mindex = 0;

    glShadeModel( GL_SMOOTH );

    glBegin(GL_TRIANGLES);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_VERTEX_ARRAY);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    for ( int i = 0; i < sizeof(face_indicies) / sizeof(face_indicies[0]); i++ )
    {
```

```

if ( !mcount )
{
    SelectMaterial( material_ref[mindex][0]);
    mcount = material_ref[mindex][1];
    mindex++;
}
mcount--;

for ( int j = 0; j < 3; j++ )
{
    int vi = face_indicies[i][j];
    int ni = face_indicies[i][j + 3];
    int ti = face_indicies[i][j + 6];

    glNormal3f(normals[ni][0], normals[ni][1], normals[ni][2]);
    glTexCoord2f(textures[ti][0], textures[ti][1]);
    glVertex3f(vertices[vi][0], vertices[vi][1], vertices[vi][2]);
}

}

glEnd();

SwapBuffers(hdc);

wglMakeCurrent( NULL, NULL );
}

```