This is a modified release of the Web3D Consortium's final report for contract award # **W81XWH-06-1-0096** to TATRC.

Contents referring to draft X3D specifications have been removed, per Web3D Consortium policy, as these have not yet been ISO ratified. Once these draft specifications have been ratified, they will be released in their complete and final form, and the public will be granted unrestricted, royalty free use of their contents.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
20-11-2007	Final	20 Nov 2006 - 20 Oct 2007

4. TITLE AND SUBTITLE

Development of an Amendment to X3D to Create a Standard Specification of Medical Image Volume Rendering, Segmentation, and Registration

5a. CONTRACT NUMBER

W81XWH-06-1-0096

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Michael Aratow, Nigel W. John, Justin Couch, Daniel Evestedt, Alan Hudson, Nicholas Polys, Richard Puk, A. Ray, Keith Victor, Qiming Wang, Afzal Godil

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Web3D Consortium
325 Sharon Park Drive, Suite 623
Menlo Park, CA 94025

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

There is currently no open standard file format for the volumetric rendering, segmentation and registration of medical imaging data. X3D is the current royalty free, ISO ratified standard for 3D data communication over networks. A volume rendering extension was added to X3D to allow 3D medical image volumes to be displayed and take advantage of the extensible functionality of this standard. Successful implementation of this extension with multiple rendering styles within a Windows based web browser plugin architecture proved its feasibility. An import/export library and mapping of the leading anatomic ontologies, SNOMED and FMA, to X3D metadata tags were also created to encourage and simplify development of 3rd party applications associated to this X3D extension. Comparisons were made with selected freely available volume rendering solutions running on multiple platforms with a development history of over 3 years. The demonstration implementation of MedX3D had a basic user interface in comparison but offered more available rendering modes. In many cases the quality of the rendering was close to that of the more mature software. A limited concurrent validity survey with one commercial software product showed MedX3D compared favorably in image display, inferior in

15. SUBJECT TERMS

X3D, standard, 3D, volumetric, segmentation, registration, medical, visualization, SNOMED, FMA, Flux, 3DView, OsiriX, ImageJ, LEONARDO

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			USAMRMC
U	U	U	UU		19b. TELEPHONE NUMBER *(Include area code)*

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Table of Contents

Introduction

The advanced radiographic technologies of today are generating hundreds to thousands of serial cross sectional or sagittal two dimensional (2D) images, which require significant time to interpret using traditional methodologies. Volumetric rendering of these images allows a more comprehensive but concise view of such large datasets, and this method of display is more intuitive as it resembles the real life appearance of the area being imaged [1]. Volumetric rendering is also creating new tools for diagnosis and treatment through such techniques as data fusion, and new alternatives to invasive procedures using virtual travel through body spaces [2]. There is currently no open standard file format for the volumetric rendering, segmentation and registration of medical imaging data. A standard three dimensional (3D) file format would provide improved universal access to 3D medical images by enabling interoperability with vendor specific Picture Archiving and Communication Systems (PACS) and 3D imaging processing and display applications. This project has developed an ISO formatted component extension to X3D (an already existing open, royalty free and ISO ratified standard for transmission of 3D data over networks) [3], to enable volumetric rendering, segmentation and registration of multimodal medical images within its specification. Additionally, an open source web browser-based example implementation running in a Microsoft Windows environment, an import/export library and a mapping between X3D and the two most popular anatomic ontologies, the Systematized Nomenclature of Medicine-Clinical Terms (SNOMED) and the Foundational Model of Anatomy (FMA) [4, 5], were also created for fostering widespread adoption and accessibility of this standard.

Please note that any content representing draft specifications of components or profiles for X3D is proprietary. This content is included in several of the Appendices (specifically Appendices B, C and F), and these pages are marked appropriately at the top and bottom. The links in these documents have been disabled. Web3D Consortium policy requires that any draft specifications are kept confidential and for viewing only by membership until formal ISO approval has been granted; then these approved specifications are made freely available to the public for download upon our website at www.web3d.org.

Task 1: Completion of MedX3D profile and Volume Rendering Extension

Section 1. Create MedX3D Profile

The volume rendering component (VRC) extends the functionality of the X3D standard, so an initial task was to define a profile of X3D, named MedicalInterchange profile, which identified specific parts of X3D that would be necessary for medical image visualization and interaction. This profile is critical to simplifying the efforts of developers to work with X3D for medical images, since X3D encompasses significant general functionality across many vertical markets including visual simulation, scientific

visualization, computer aided design, instruction (manuals, education) and collaborative virtual environments. Through interviews with end users and vendors within the Digital Imaging and Communications in Medicine (DICOM) community, several use cases were developed. They generally fell into the categories of education, surgical or procedural planning and enhanced accessibility (see Appendix A). The features and functions implied by these use cases were then translated into a definition of the MedicalInterchange profile (MIP) (see Appendix B). An open, royalty free and ISO ratified standard for 3D images enables these scenarios because it lowers the cost of ownership, enables interoperability and directs vendors to compete on functionality and features since there is no monopoly on content expression. The MIP obviously includes nodes related to geometry and appearance, with limited nodes for collision detection, navigation, text and annotation, providing a much simpler path towards implementation than if the entire X3D specification had to be considered.

Section 2. ISO Format Ready Volume Rendering Component

The VRC to X3D was created after a thorough search of volume rendering styles in the literature was undertaken. For example, refer to the new medical visualization text by Preim and Bartz [6]. It was decided that the focus of the specification was to add the most common and currently used rendering styles, including the Gooch shading model of two-toned warm/cool coloring [7]. There was also a priority placed on simplicity, both to benefit people making an implementation of the specification and for the user making content using X3D. Although a limited number of rendering nodes were initially declared, the VRC will continue to evolve and more advanced/general rendering methods can be added later at a higher support level. The X3D specification also provides a well-defined extension mechanism called Components that allows vendors to experiment with additional functionality beyond the explicitly rendering styles specified. This mechanism can be used to prototype newer, less well tested technologies, such as a more general framework capable of expressing multi-dimensional transfer functions, for inclusion in later revisions of the VRC.

Valuable discussions occurred between the vendor involved with writing the VRC (Yumetech) and the vendor involved with implementing the VRC within the MedX3D browser (SenseGraphics) in creating this draft component specification. The ISOSurfaceVolumeData node took an especially large amount of time to determine its description such that it would not only be relevant for medical images, but also complex scientific visualizations. The VRC to the X3D specification can be found in Appendix C.

Section 3. Expert Review of Volume Rendering Component

Upon drafting of the VRC, two experts in the field of computer graphics, but not members of the Consortium, were asked for their comments. The first evaluator's comments are in Appendix D (the Word document referred to in his report was left out as it is redundant information). The second evaluator's comments are also contained in Appendix D.

The reviewers gave the VRC draft a 4 out of 5 rating in quality, understanding that compromises had to be made to keep this initial version simple yet as comprehensive as

possible. There were no major concerns about the VRC's approach and both reviewers noted areas where some language was vague or misstated.

Section 4. Documentation for Interfacing to DICOM

As all advanced imaging modalities output to the DICOM standard [8], a guiding document was created to assist developers in interfacing to the DICOM file format from an implementation of the VRC and MIP (see Appendix E). This document provides an overview of the DICOM standard, a definition of important DICOM terms, DICOM storage methodology, aspects most relevant for a X3D implementation using DICOM and toolkits to help with parsing and conversion of DICOM compliant image files.

Section 5. Annotation Component

It was realized early on in the project that to give the most functionality to developers and end users for labeling regions of interest in volumetric shapes (or anything in a virtual environment), an additional component to X3D would need to be developed. Therefore, an annotation component draft was created and is included in Appendix F. This is a work in progress and is outside the scope of the Statement of Work, but will not only benefit the MIP, but many other vertical markets using X3D.

Task 2: Mapping between X3D and SNOMED/FMA

The overall goal of this task was to provide a method for semantic medical data to be incorporated into an X3D scenegraph via the use of the Metadata node set. The knowledge bases targeted were FMA and SNOMED [4,5], the most commonly used anatomical ontologies. Work consisted of five phases or sections that covered: obtaining FMA and SNOMED information, finding appropriate representations of that information in X3D Metadata, creating transformation mechanisms for the information, evaluating their similarities, and finally showing examples of the metadata in use. This methodology is divided into five different sections detailing how each of these areas were completed.

This work shows that it is possible to integrate both FMA and SNOMED information individually into an X3D scenegraph with a lossless transformation. It is theoretically possible that both FMA and SNOMED transformation could be combined together in one single mapping tool handling both data sources, but with current resources this was outside the scope of this contract. In addition, the separation of tools helps to illustrate and clarify their differences regarding structure and terminology.

The results of this project may constitute Recommended Practice for the Web3D Consortium's Medical Working Group and inform the development of semantically-integrated interactive 3D applications (i.e. anatomy browsers and imaging (DICOM) tool vendors). A common scenario would be to use these conventions to semantically 'tag' segmented anatomical structures for storage or delivery as an X3D environment.

The general nature of the conversion is to have a MetadataSet node contain items with multiple attributes and to use a metadata string or integer for the information. The toolset we provide is based on the following simple premises, used as conventions:

1. MetadataSet nodes refer to their sibling Transform node, where the object's shape geometry may be specified. A sibling Group node may be instantiated for parts or subdivisions of the referent object. This allows larger containing structures or anatomical systems to be easily accessible programmatically and additional detail accessible when needed
2. The MetadataSet node is instantiated with its source specified as the reference field (e.g. FMA,. SNOMED); its children are typically MetadataString nodes specifying its attributes and its relationships to other entities in the source ontology
3. Unique identifier names of source entities (integers) are prepended with an 'm'. This allows result data to conform to the Web3D scenegraph identifier convention (DEF); to cross-reference corresponding entities in the scene-graph or to programmatically access named nodes, one must remove this first character ('m') and compare it with a MetadataSet node's name="" attribute.
4. If source information is of type Integer, a MetadataInteger node is instantiated
5. If source information is of type Boolean, a MetadataString node is instantiated with a value of true or false.

The contents of the deliverable tarball implementing these conventions are listed in Appendix G, Section A. The tarball is available at :
http://snoid.sv.vt.edu/~anray2/x3dMetadata.tar.gz

Section 1. Obtain working copies of FMA / SNOMED

1.a FMA

We obtained information about FMA by going to their website, following their instructions of downloading the MySQL data, installing the data into a local MySQL database, installing Protege (an ontology viewer / editor), and installing PHPMyAdmin for investigating the MySQL database via web pages and forms. The website for FMA and a set of instructions to duplicate our work on localhosts can be found in Section B of Appendix G.

An example PHPMyAdmin viewing account to the FMA MySQL can be found online at:

http://snoid.sv.vt.edu/npolysWWW/phpMyAdmin/index.php

user: web3d password: web3d

1.b SNOMED

Currently, there is very little information publicly accessible concerning SNOMED. The main source of information is http://www.snomed.org but it is currently in transition. One of the websites linked from the main site contains the XML Schema and an example SNOMED XML file. This can be found here:

http://www.ihtsdo.org/our-standards/technical-documents/#c586.

In the future more information and examples will hopefully be provided.
The main difficulty in achieving these tasks dealt with finding information about SNOMED and the system configuration of Protégé for FMA. Once this was finished, we moved to the next step of transforming the information from these packages into a format that could be instantiated into a X3D scenegraph in a lossless manner.

Section 2. Determine a mapping between FMA -> X3D

FMA has information at many different levels. They are both theoretical and practical. At a theoretical level FMA is composed of a body that has many different parts and each part has a parent / child relationship and several different pieces of information associated with it. A way to map this into X3D metadata would be to have a metadata set for each part, have metadata strings and integers to contain information about that part, and to provide the names of the other parts that it is associated with. This will allow for the relationships and information in FMA to be represented.

The format of the information of FMA comes in two different forms. The first is raw SQL information (selected using a 'produce xml' option). The second are the forms that Protege can export. Protege itself allows for browsing of the FMA and exporting in several different formats.

2.a FMA via SQL Queries

MySQL can deliver database query results as XML. The transformation can therefore be done through an XSLT file. We implemented and included it in the XSLT folder in the tarball (FMA.xslt). The program xsltproc can be used to apply the stylesheet to output files. Examples of an FMA SQLrecord and how this can be transformed into target X3D metadata can be found in Appendix G, Section B. Figure 1 shows a screenshot of a proof of concept demonstration using the commercially available X3D browser from Octaga (www.octaga.com) where placing a mouse over the liver shows the FMA output from MySQL.

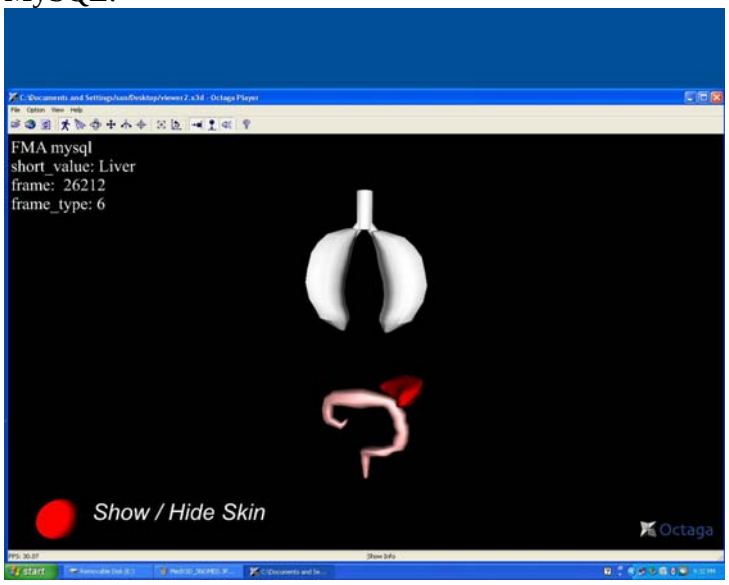


Figure 1: FMA SQL query displayed when mousing over the red liver (posterior view of body)

2.b FMA information via Protege

Another form of information that we have used is to produce X3D metadata in HTML output from Protege. Protege allows for the export of information in several different forms; however the only complete export format that contains all of the information in the database is the 'HTML output' option. All of the other options (RDF / OWL / XML are a few) require database lookups to completely fill in all of the information. An example Protege HTML output page can be found in Appendix G, Section B or in the tarball provided with the report:

http://snoid.sv.vt.edu/~anray2/parser/Upper+lobe+of+lung.html

Part of the work for providing a pathway for FMA metadata to be incorporated into X3D was to build a digital library of FMA information. A C++ program was developed to convert these HTML files into X3D metadata. From a standards prospective, an XSLT file is preferable to a C++ implementation, but due to the nature of the source data in this case (HTML), C++ was the best solution for transforming to X3D metadata. Instructions on how obtain, build and run this program and an example of the metadata produced by this program are in Appendix G, Section B. Figure 2 shows a screenshot of a proof of concept demonstration using the Octaga X3D browser where placing a mouse over the lungs shows the FMA output from the C++ parser.

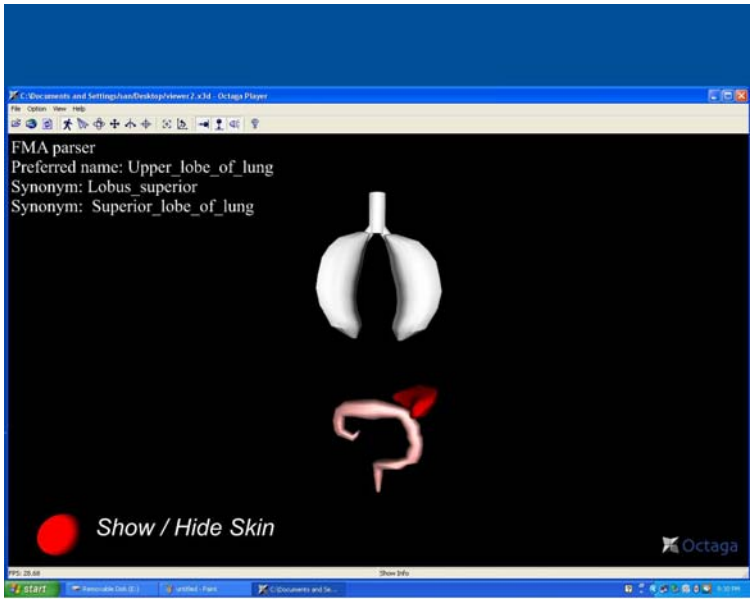


Figure 2: FMA direct parser query displayed when mousing over the lungs (posterior view of body)

Section 3. Determine a mapping between SNOMED -> X3D

SNOMED is much simpler to convert into X3D metadata than FMA. SNOMED already comes described by way of XML tools, so converting it to X3D metadata is simply transforming the 'styling' XML file. This is done via a XSLT. The instructions for how to obtain and run this file are in Appendix G, Section C.

We generated documentation for the SNOMED Schema with xnsdoc:

http://snoid.sv.vt.edu/~anray2/snoMed/

The complete input and output for the SNOMED knowledgebase is rather extensive so only a small part will be used to explain the strategy for creating X3D metadata out of SNOMED information. In general, the SNOMED data is grouped into different elements that have several different attributes in each tag. The general strategy for converting this into X3D metadata was to create an overarching metadata set that contained the same structure of the existing format by using sets with the attribute name that holds the old elements name. After this, all of the attributes are incorporated as metadata strings with name fields being the attribute and the value field being the value of the attribute. An example of SNOMED input and the transformed XML into X3D metadata can be found in Appendix G, Section C.

The SNOMED XSLT file that converts the mapping is quite complex (>600 loc). For performance and future maintenance, a C++ program would be a much better fit. Due to the original work plan, an XSLT file was produced to meet the deliverable, but if the SNOMED standard is changed, it may be simpler to write a C++ program instead of modifying the XSLT file. Figure 3 shows a screenshot of a proof of concept demonstration using the Octaga X3D browser where placing a mouse over the intestine shows the SNOMED output.

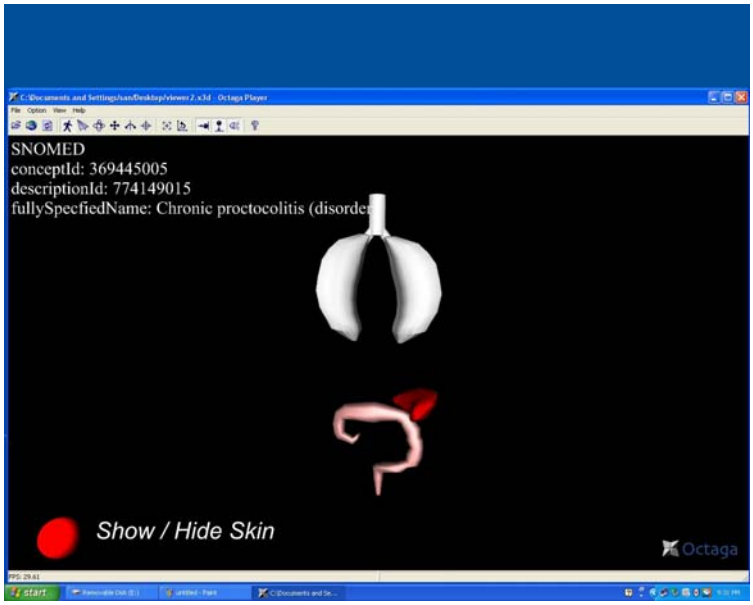


Figure 3: SNOMED queery displayed when mousing over the intestine (posterior view of body)

Section 4. Determine a mapping between FMA / SNOMED

SNOMED has a much richer design space than FMA does. However, it has the notion of relationships that can encompass the part / partOf relationship that FMA uses. Thus, it is theoretically possible to map the two different ontologies. A more thorough investigation into how the two ontologies can correspond can be found elsewhere [5]. However, this requires foresight into designing the SNOMED data, because FMA is completely fixed and cannot be changed, whereas SNOMED can be manipulated when particular data files are being created. If a SNOMED file is created that contains the correct names and identification numbers for specific part relationships in FMA, then it would be possible to

integrate both SNOMED and FMA metadata. The other option would be to include a Grouping MetadataSet node that would contain a FMA MetadataSet node and a SNOMED MetadataSet node. If this were done, then the existing conversion tools could be modified slightly to accommodate this information.

Section 5. Example usage of metadata

There are many possibilities for applying this work in X3D medical applications. Some specifics of the node structure in the delivery scenegraph will depend on the application requirements. The tools and recommended practice we provide can be widely adapted for these needs. For example in X3D, the interface for an anatomy browser could be created with menus driving the visibility or highlights of shapes with a Switch node; referent shapes can also be Inlined. Alternatively, a program could walk the scenegraph, apply some logic, and display specific metadata information as annotations or labels.

We include an example of the FMA and SNOMED information from this report in the following files (see Figures 1-3 for their output):

- body_ifs.x3d – An IndexedFaceSet from NIST constituting an androgynous human skin
- med_example.x3d – A simple MedX3D scenegraph structure with shapes and metadata conforming to our recommended practice; it includes a mix of FMA and SNOMED information for skin, lung, liver, colon
- viewer2.x3d – Adds a simple interactive interface to some of the data from med_example.x3d and demonstrates a simplistic anatomy / ontology viewer

Task 3: Import/Export Library

The goal of this task was to create a software library that application developers can use to aid in the development of software which will interact with MIP content.
In addition to the ability to import and export content, the library was required to support an API which will allow developers the ability to interact with the scenegraph. Developers must be able to generate content programmatically which can subsequently be exported. The API was also required to be able to interrogate the content that has been imported.

The scope of the X3D content that this library must support is the MedicalInterchange profile (MIP), which is clearly defined in Appendix B.

The development of the Import/Export Library utilized the Flux X3D-based open source engine. Source code for the Flux engine, along with this Import /Export library, can be accessed from: http://sourceforge.net/projects/flux (there will be some lag time before updates).

The API supported by this library is very closely aligned with the Scene Application Interface (SAI), which is the mature API for X3D. For a thorough background on the X3D SAI, please see:
http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/

COM was used for the library infrastructure. Since this library does not contain a runtime environment, there are many methods in the SAI that do not apply, therefore, only a subset of the SAI is included in this library. Also, the requirements of this project necessitate a few additional methods be added to the API, the most important of which is the ability to export the scenegraph. Details of the API, including installation instructions, tips for getting started and the API reference guide can be found in Appendix H.

The installation also includes two test harness applications. One in Visual Basic, and one in C++. The C++ test harness application shows how the API can be used to import and interrogate an X3D file by displaying the node hierarchy of the contents in a GUI tree window. The test harness then programmatically adds content to the scene, and exports the result, thus illustrating the ability to programmatically generate X3D content.

Task 4: Browser Implementation

Section 1. MedX3D Profile and Volume Rendering Extension Browser Implementation

A Windows based web browser plugin and standalone application were created which implemented both the node definitions in the MIP (Appendix B) and a subset of the VRC (Appendix C). Specifically within the VRC, the basic rendering style, Opacity Map, and advanced rendering styles: Boundary Enhancement, Silhouette Enhancement, Edge Enhancement, Maximum Intensity Projection, Segmented Volume, Tone Mapping, Cartoon Style, Isosurface Volume and Composed Volume (user designated combination of any of the available rendering styles) were also implemented. These represented eight additional advanced rendering styles that were implemented beyond the requirements of the Statement of Work. The only rendering style not implemented in the VRC was the Stipple Volume rendering style. Appendix I illustrates many of the volume rendering styles implemented in the MedX3D browser.

The rendering engine is based on the H3D API (www.h3dapi.org), a dual commercial and GPL (open source) licensed software product that is a development platform for multi-sensory applications created by SenseGraphics (www.sensegraphics.com). This API uses X3D, OpenGL, C++, Python as well as leveraging haptic technology from SensAble. Rendering is based on GPU ray casting utilizing the GL Shading Language and 3D textures; reflections or refractions are not simulated, leading to better performance.

A difficulty during the browser development actually involved problematic drivers for ATI graphic processing units (GPUs). Although the MedX3D browser performed well using GPUs from NVIDIA, no display output was achieved with ATI GPUs. The problem was isolated to the ATI driver implementation of the GL Shader Language standard. SenseGraphics, the vendor who was granted the subaward for MedX3D browser implementation, worked extensively with representatives from ATI to fix the problem. Further iterations of the ATI drivers were successful in enabling the MedX3D browser to create an image, but only using limited rendering styles. Work continues to enable all rendering styles using ATI GPUs within the MedX3D browser.

Section 2. Picking

Picking is defined as limited testing of arbitrary object collision. In earlier versions of the standalone application, this was demonstrated by the rendered volume having a cube outline surrounding the volume identifying the bounding box being used for collision detection, as seen in Figure 4 below.

This was removed in the final version as it was felt that navigation would be too limited for the user to explore the volume dataset. Picking can easily be reactivated by making minor programmatic changes in the source code and recompiling.

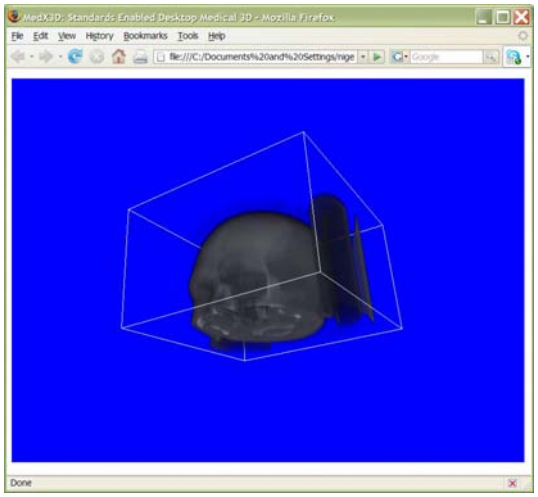


Figure 4: Bounding box visible surrounding volume in earlier version of MedX3D browser

Section 3. DICOM loading

The standalone application is able to load a single DICOM file, any DICOM file set within a folder (by clicking on only a single file of the set) or .raw files. The browser plugin can only load X3D files.

A problem encountered with this subtask was that certain DICOM datasets would not display in the browser. This seemed to be occurring with MRIs and CTs of the chest and any body section inferior to this, although there were isolated cases of problems with MRIs of the brain. The DICOM loader in the MedX3D browser uses the DCMTK DICOM-Toolkit (http://dicom.offis.de/dcmtk.php.en) , the same as used in both OsiriX

and 3DView, two of the software products used in the comparison testing (see below) and since these viewers had no difficulty displaying the same images, there is obviously a problem in the DICOM loading of the MedX3D browser which needs to be addressed.

Section 4. Comparison Testing

As part of the assessment of the utility of the MedX3D browser at this point in its development, comparisons were made between its output and that of other desktop computer based volume rendering software products that are freely available for download and are relatively popular. Four files were chosen for this comparison: 1) MRI of inner ear in DICOM format, 2) MRI of head and neck in DICOM format, 3) MRI angiogram of brain in raw format and 4) CT of the abdomen in DICOM format. Although the MedX3D browser can be used both as a standalone application and a web browser plugin, testing was performed in its standalone version as the other applications to which it was compared are not browser plugins.

For Windows based PC's, 3DView from RMR Systems Limited was chosen due to its compatibility with a wide range of graphics cards and its use of the OpenGL API for rendering (http://www.rmrsystems.co.uk/volume_rendering.htm); this is the same API on upon which MedX3D is based. Only opacity map rendering was used: its variables were adjusted in MedX3D and 3DView had its transfer function adjusted both as necessary for the clearest picture.

The Windows based PC test machine was running Windows XP with Service Pack 2, a monitor running 1280 X 1024 resolution at 32 bit color, a NVIDIA 7300LE graphics card with ForceWare driver version 163.21, DirectX 9.0c and 3DView version 1.2.

For Mac based PC's, OsiriX (http://www.osirix-viewer.com/), an open source program developed in 2004 exclusively for the Apple OS X operating system and using the OpenGL API for rendering, was chosen because of its widespread international popularity, free availability and advanced feature set. OsiriX is not only a DICOM viewer, but also acts as a PACS, integrating functionality that is usually found in two separate products (although the trend is increasingly towards integration of these two systems). OsiriX also has the capability of allowing developers to create plugins to increase its functionality. Again, only opacity map rendering was used for the rendering style for comparison purposes with the MedX3D browser.

The Mac based PC test machine was running Mac OS X Version 10.4, a monitor running at 1920 X 1200 resolution and 32 bit color, a NVIDIA GeForce 6800 Ultra graphics card and OsiriX version 2.6 32-bit.

For Linux based PC's, Image J was used. ImageJ is an open source, public domain, Java-based image processing program developed in 1997 at the National Institutes of Health. It is extensible via plugins and recordable macros, and can be run as an online applet or a standalone application on any computer with a Java 1.1 or later virtual machine. Image J

has versions for Microsoft Windows, Mac OS, Mac OS X, Linux, and the Sharp Zaurus PDA environments.

The Linux based PC test machine was running SUSE 10.3, Sun Java 1.5.0_13, a monitor running 1280 X 1024 resolution and 16 bit color, an ATI Radeon X1900XT graphics card with 512MB and ImageJ version 1.38.

MRI Inner Ear—DICOM format dataset

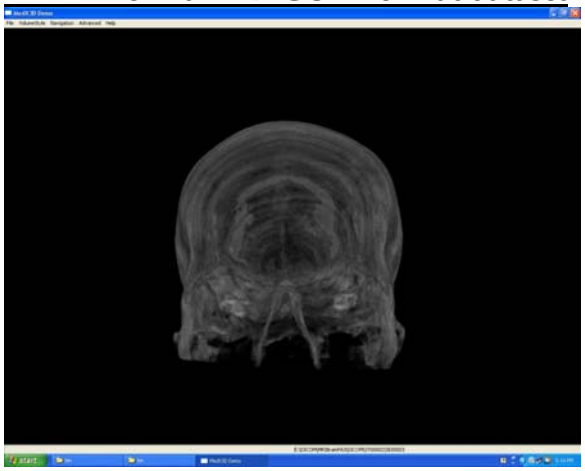


Figure 5: MedX3D standalone application displaying front view of MRI of the head emphasizing inner ear components

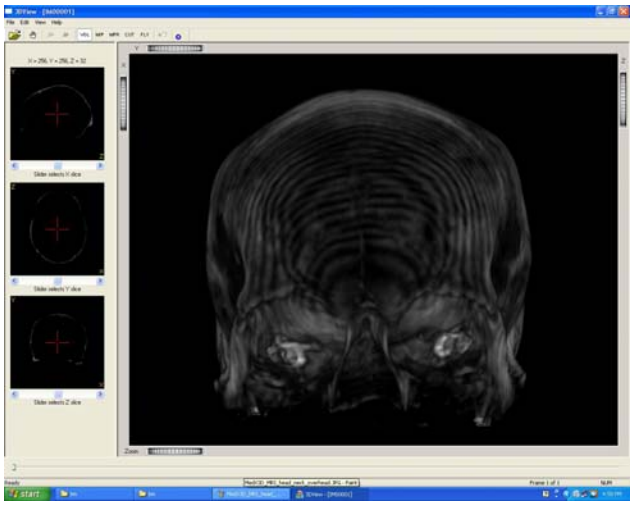


Figure 6: 3DView application displaying front view of MRI of the head emphasizing inner ear components

Figure 7: OsiriX application displaying front view of MRI of the head emphasizing inner ear components

Figure 8: ImageJ application displaying front view of MRI of the head emphasizing inner ear components

Figure 9: MedX3D standalone application displaying inferior view of MRI of the head emphasizing inner ear components

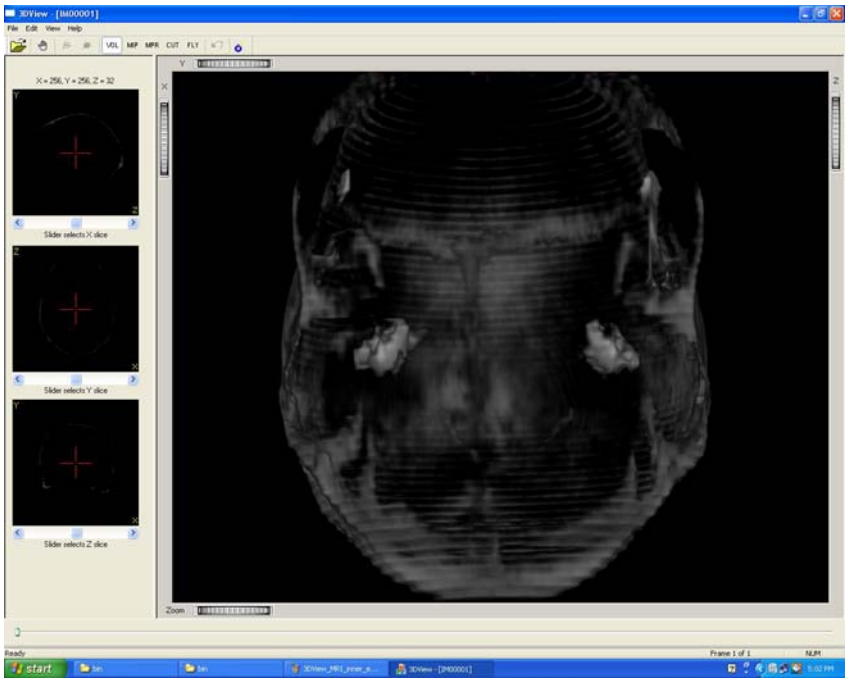


Figure 10: 3DView application displaying inferior view of MRI of the head emphasizing inner ear components

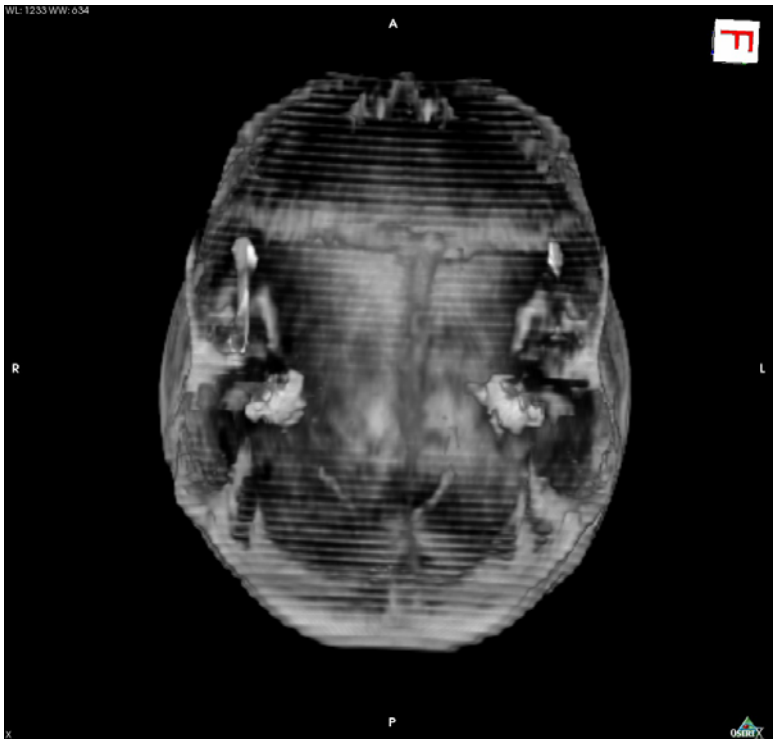


Figure 11: OsiriX application displaying inferior view of MRI of the head emphasizing inner ear components

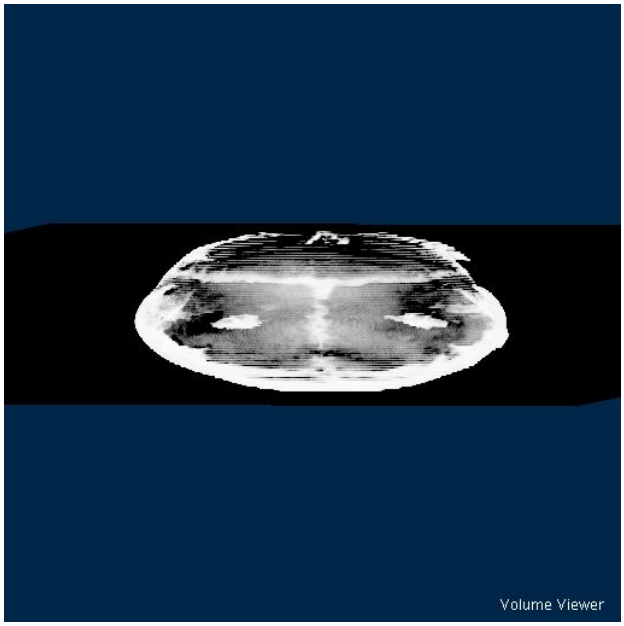


Figure 12: ImageJ application displaying inferior view of MRI of the head emphasizing inner ear components

3DView's and OsiriX's images in this dataset are clearer and inner ear structures are more easily discernable than in MedX3D or ImageJ. ImageJ is demonstrating some distortion of the dataset in the anterior posterior direction.

MRI Head and Neck—DICOM format

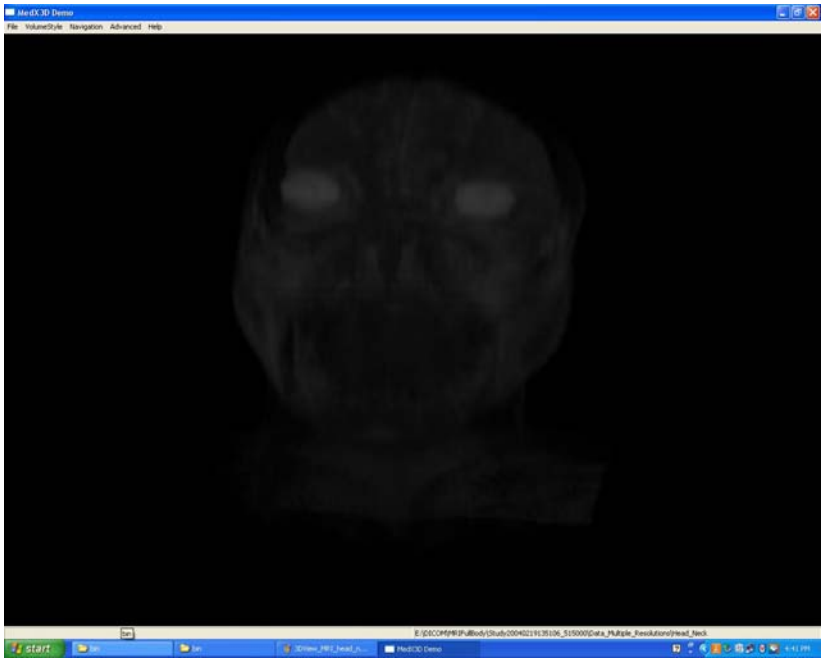


Figure 13: MedX3D standalone application displaying frontal view of MRI of the head and neck

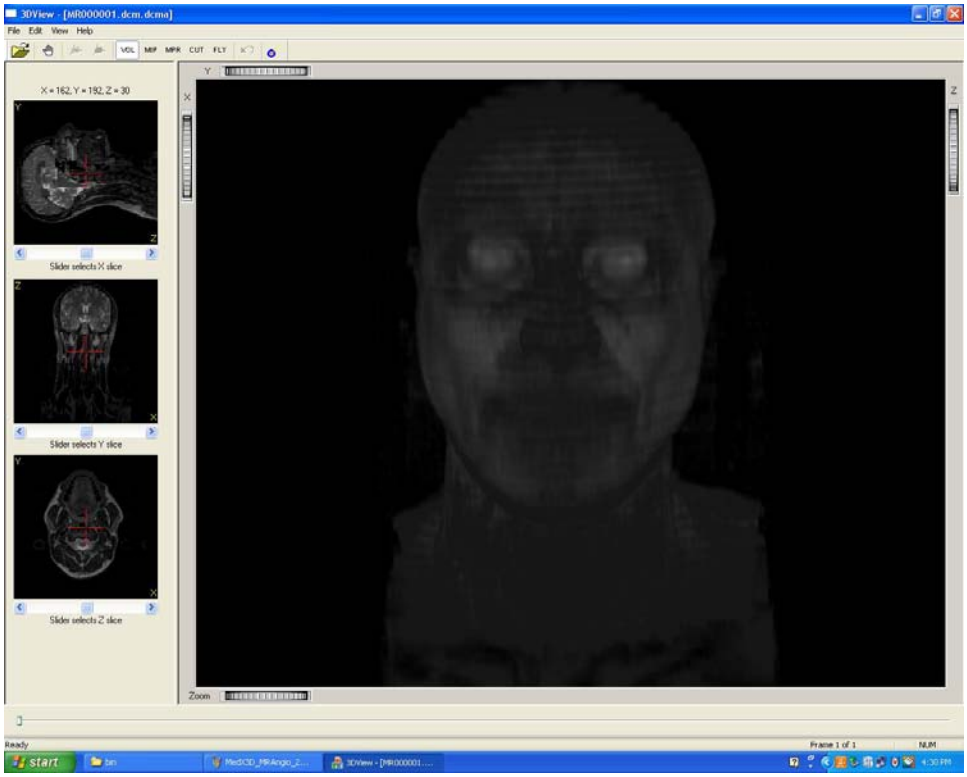


Figure 14: 3DView application displaying frontal view of MRI of the head and neck

Figure 15: OsiriX application displaying frontal view of MRI of the head and neck

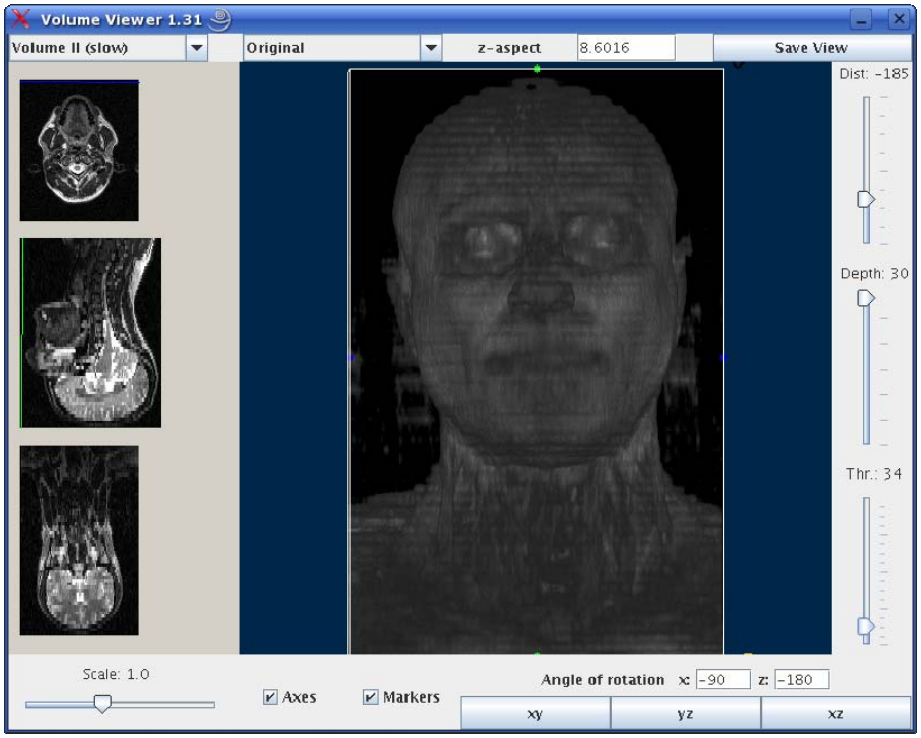


Figure 16: ImageJ application displaying frontal view of MRI of the head and neck

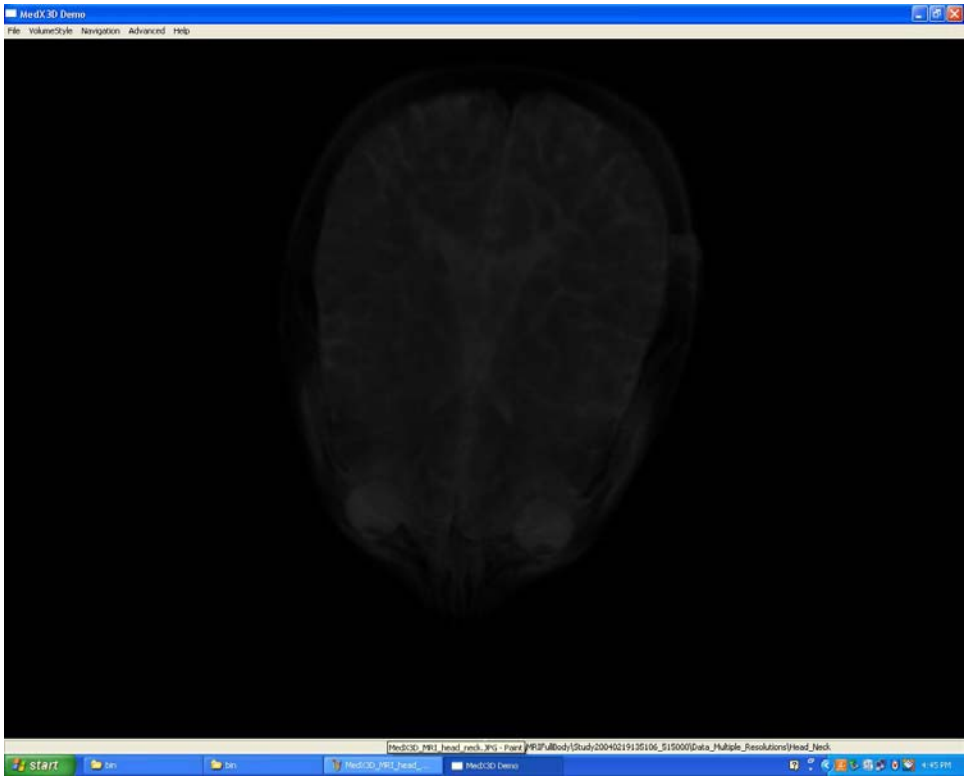


Figure 17: MedX3D standalone application displaying overhead view of MRI of the head and neck

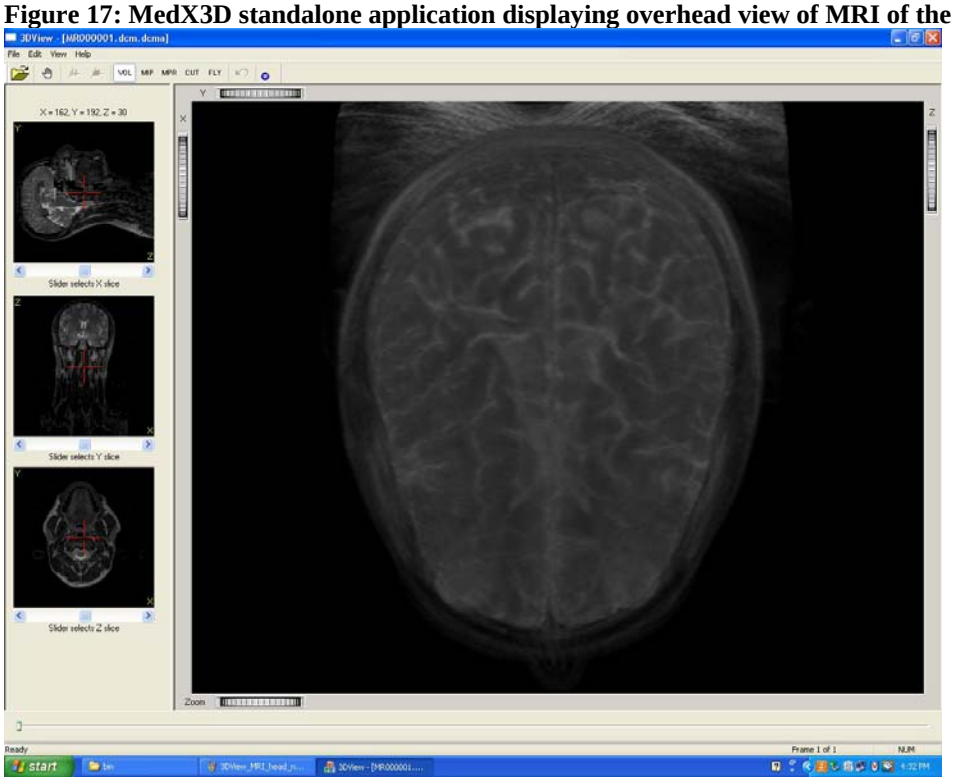


Figure 18: 3DView application displaying overhead view of MRI of the head and neck

Figure 19: OsiriX application displaying overhead view of MRI of the head and neck

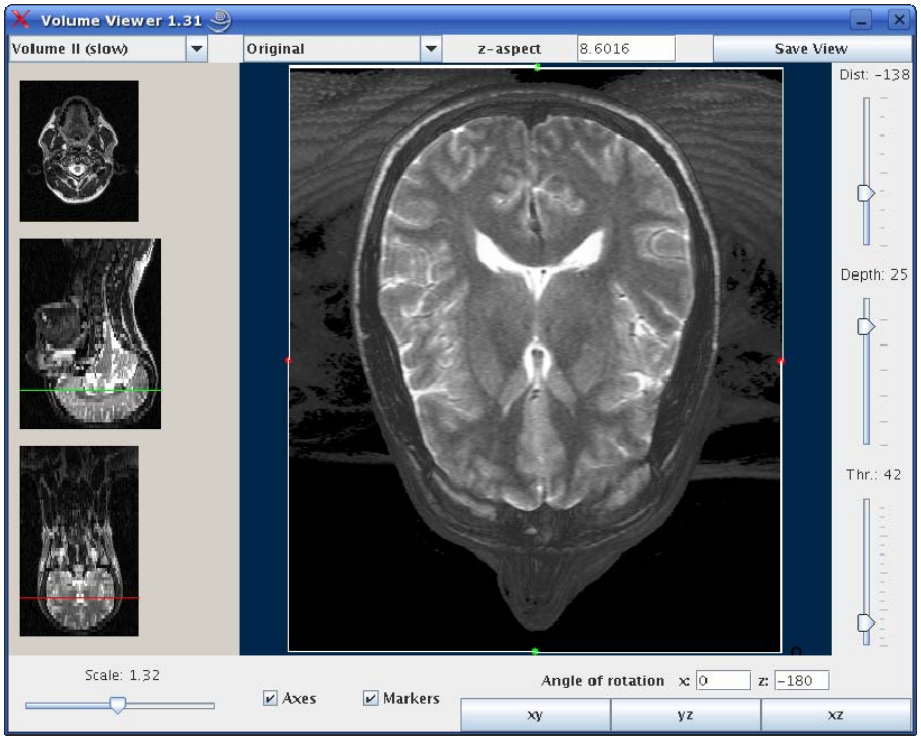


Figure 20: ImageJ application displaying overhead view of MRI of the head and neck

OsiriX has the most details and contrast and is followed closely by ImageJ. OsiriX and ImageJ have higher intensity images than 3DView and MedX3D.

MRI Angiogram—Brain raw Format

Figure 21: MedX3D standalone application displaying frontal view of MRI angiogram of the cerebral circulation

Figure 22: MedX3D standalone application displaying frontal view of MRI angiogram of the cerebral circulation

Figure 23: OsiriX application displaying frontal view of MRI angiogram of the cerebral circulation

Figure 24: ImageJ application displaying frontal view of MRI angiogram of the cerebral circulation

Figure 25: MedX3D standalone application displaying overhead view of MRI angiogram of the cerebral circulation

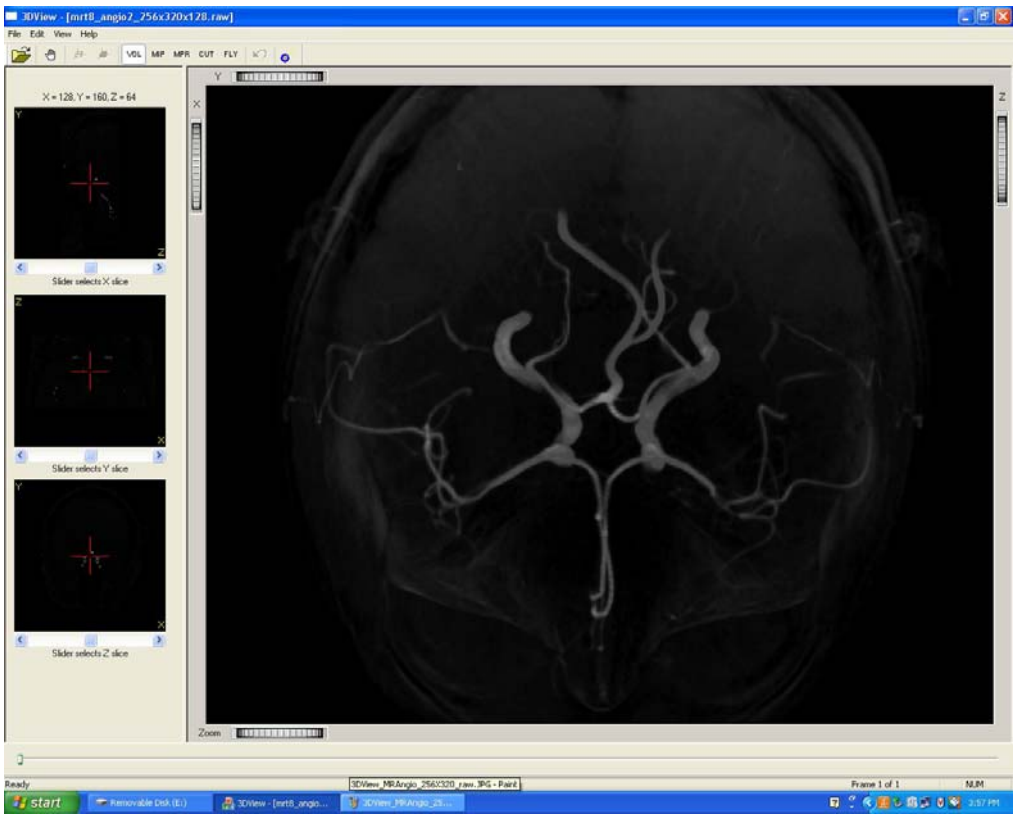


Figure 26: 3DView application displaying frontal view of MRI angiogram of the cerebral circulation

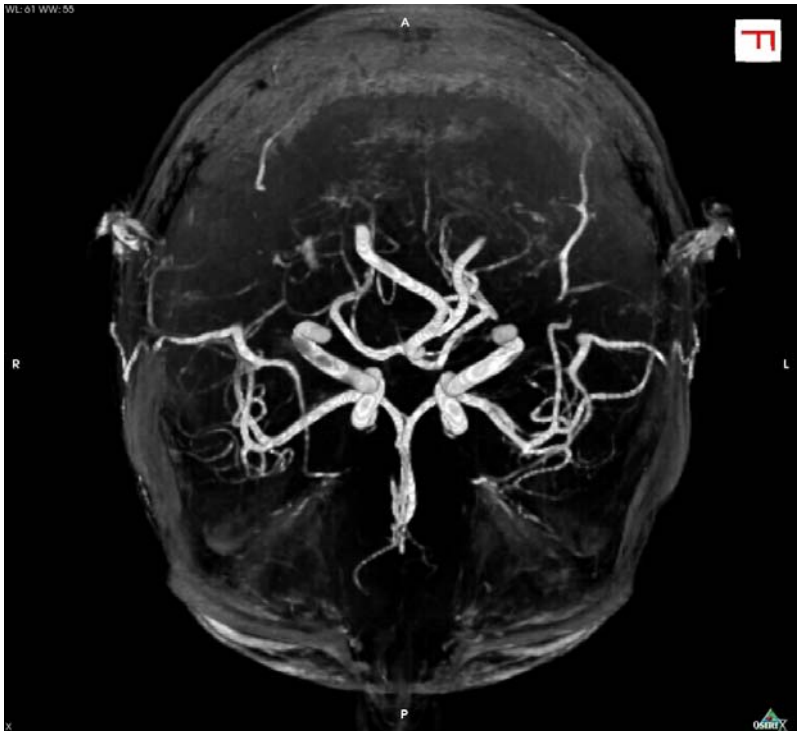


Figure 27: OsiriX application displaying frontal view of MRI angiogram of the cerebral circulation

Figure 28: ImageJ application displaying frontal view of MRI angiogram of the cerebral circulation

ImageJ provides the highest detail and contrast followed by OsiriX. 3DView has higher contrast vasculature than MedX3D, but MedX3D captures more detail in vascular structure than 3DView.

CT Abdomen—DICOM format

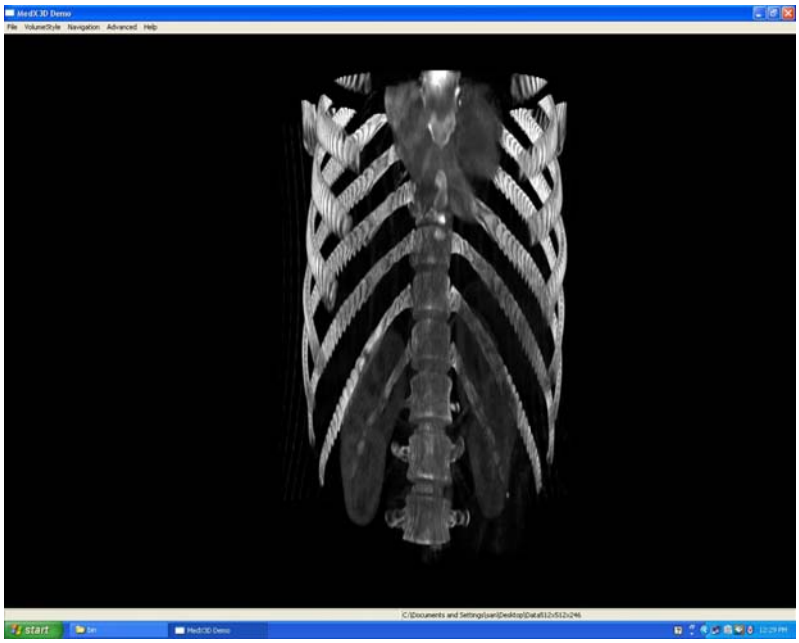


Figure 29: MedX3D standalone application displaying frontal view of a CT of the abdomen

Figure 30: 3DView application displaying frontal view of a CT of the abdomen

Figure 31: OsiriX application displaying frontal view of a CT of the abdomen

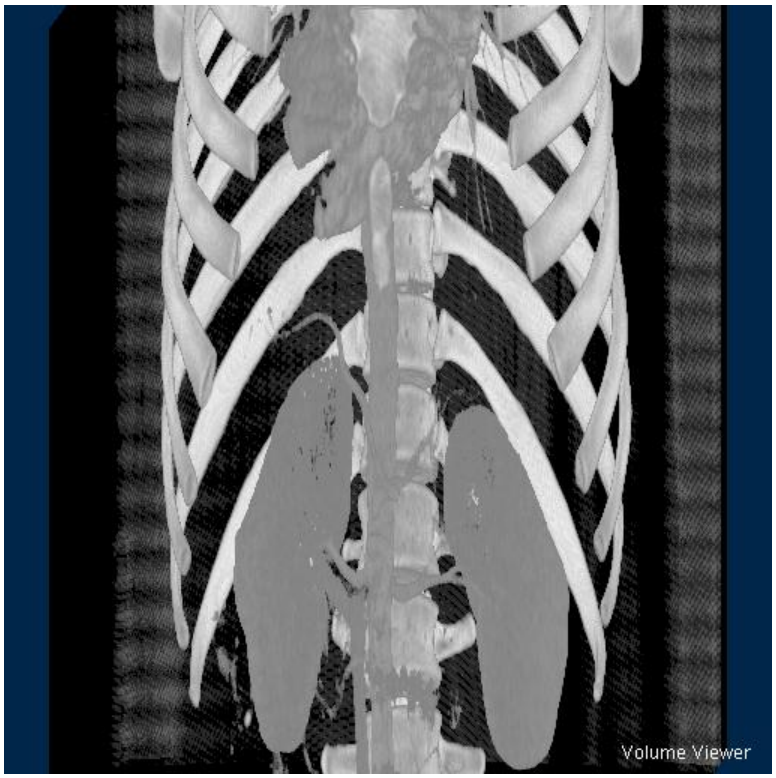


Figure 32: ImageJ application displaying frontal view of a CT of the abdomen

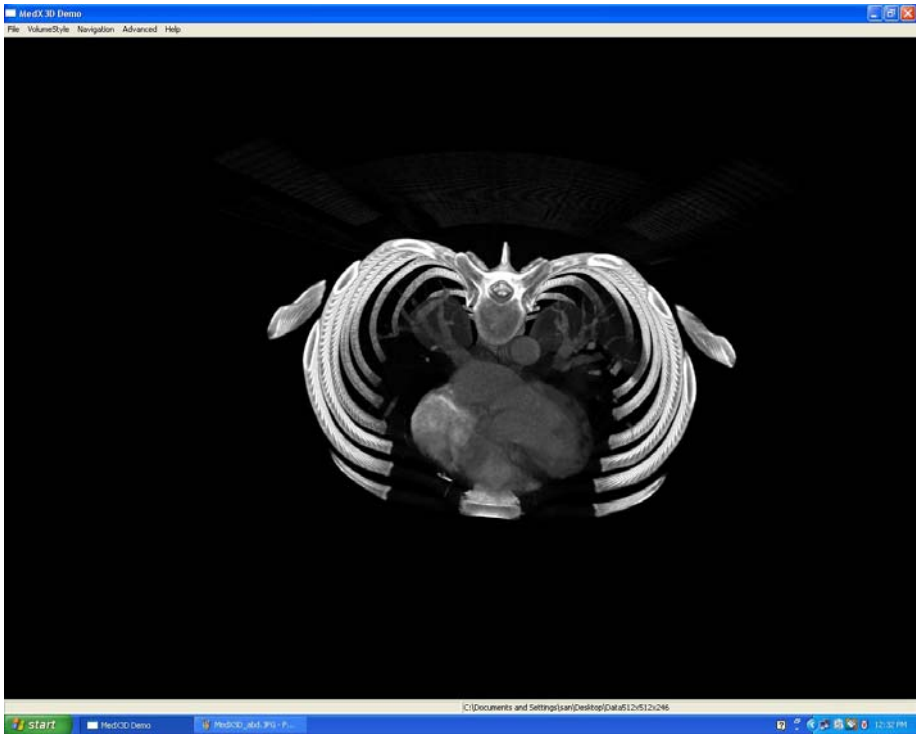


Figure 33: MedX3D standalone application displaying overhead view of a CT of the abdomen

Figure 34: 3DView application displaying overhead view of a CT of the abdomen

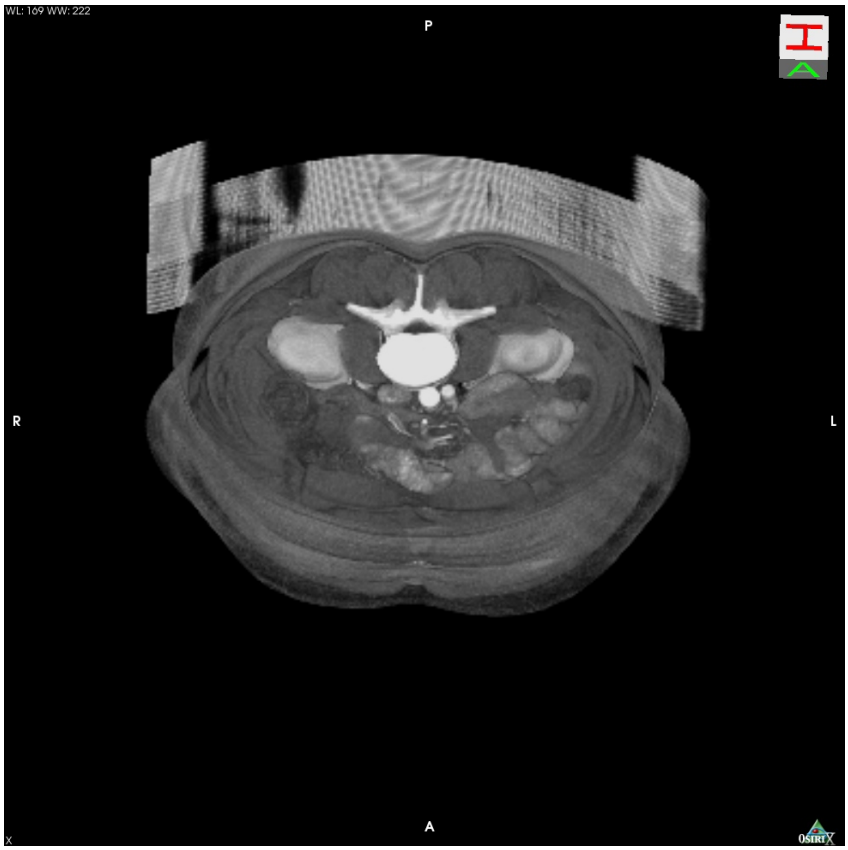


Figure 35: OsiriX application displaying overhead view of a CT of the abdomen

Figure 36: ImageJ application displaying overhead view of a CT of the abdomen

ImageJ and MedX3D seems to display the dataset with some vertical distortion, whereas OsiriX and 3DView display the data with normal proportions. 3DView is able to accentuate the kidneys better, but MedX3D and ImageJ are able to reveal finer contrast, such as pulmonary vessels.

OsiriX and ImageJ provided the best overall image quality in this comparison test and are good benchmarks for freely available medical image volume rendering software. OsiriX is a standalone application and cannot operate as a web plugin, nor is it based on a royalty free, open standard, ISO ratified file format, in contrast to MedX3D. ImageJ is also not based on a standard, but it can run as an applet within a web page.

As expected for products that have been in development for at least 3 years, ImageJ, OsiriX and 3DView display better quality output than MedX3D. OsiriX, ImageJ and 3DView also boast much more image manipulation capabilities, such as segmentation and GUI accessible isosurface creation functions, than MedX3D. MedX3D does have the advantage of a 10 year revision history not focused on medical image rendering, but on 3D environments on the web. This gives the advantage of more applications for end users and interoperability of data between applications.

MedX3D Concurrent Validation

Section 1. Introduction and Methods

A statistically significant concurrent validation study to compare the MedX3D browser against the gold standard for 3D medical visualization was outside the scope of this project. However, it was possible to make some initial comparisons between the MedX3D browser and an example of the commercial 3D environments currently used in hospitals.

The Radiology Department at the Royal Liverpool Hospital in the UK agreed to take part in this exercise. A MRI scan was taken of a healthy volunteer, using the Philips Achieva 1.5T scanner located at the hospital. Note that ethical approval would have been needed to use an existing patient's data.

Scan Details (extract):

Acquisition Time: 141829.40
Image Time: 141829.40
Modality: MR
Manufacturer: Philips Medical Systems
Institution Name: Royal Liverpool Hospital
Study Description: BRAIN
Series Description: T2/TSE
Manufacturer's Model Name: Achieva
Scanning Sequence: SE
Sequence Variant: SK
Slice Thickness: 3.0
Magnetic Field Strength: 1.5
Spacing Between Slices: 3.29999995231628
Rows: 512
Columns: 512
Pixel Spacing: 0.44921875\0.44921875
Pixel Aspect Ratio: 1\1
Bits Allocated: 16
Bits Stored: 12
Window Center: 1023.0
Window Width: 2047.0

The voxel resolution of the scan data is 512 x 512 x 42, using an axial acquisition plane. Thumbnails of some of the scanned images can be seen in Figure 37.

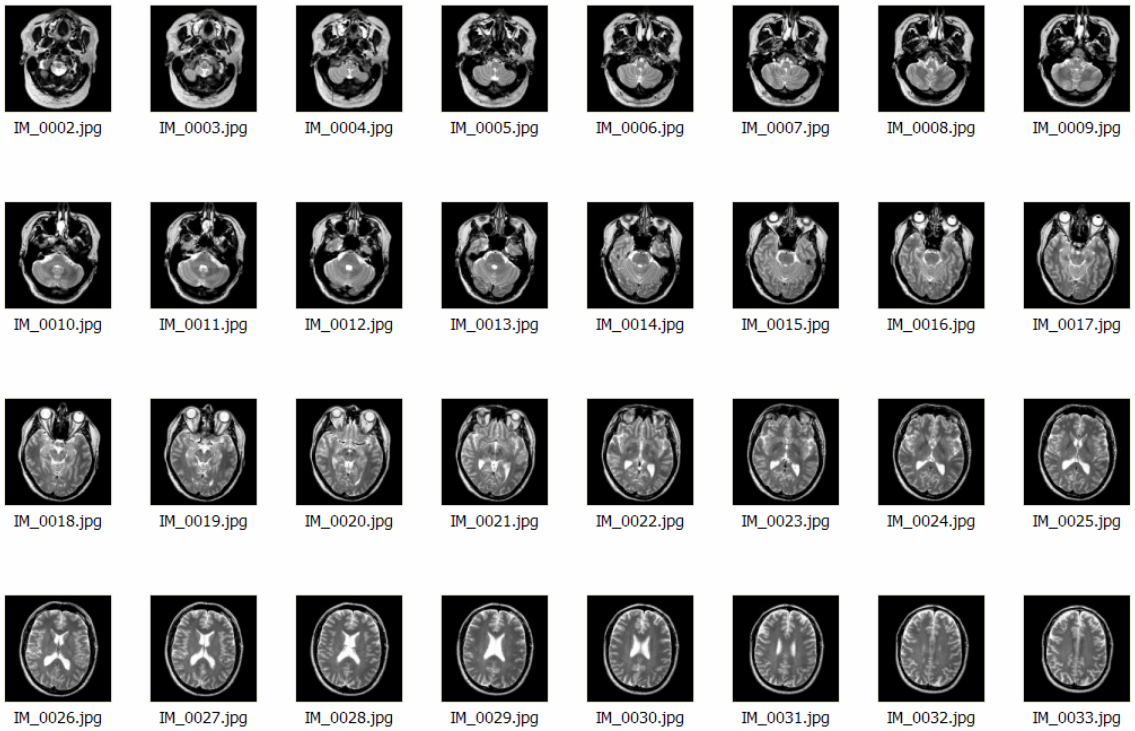


Figure 37: 2D Images from Test MR Scan (here converted to jpg format)

Most of the 3D visualization of medical scans at the hospital takes place on a Siemens' LEONARDO post-processing workstation connected to the hospital PACS. The LEONARDO is based on a PC platform, and supports the following 3D methods: MIP, MPR, SSD, and VRT. We use the VRT (Volume rendering textures) method in our comparison with MedX3D.

The MedX3D software was installed on a Dell Precision M65 laptop, containing an NVIDIA Quadro FX 350M graphics card. It was not possible to connect the laptop to the hospital PACS for security reasons. The MR scan data was therefore copied to a CD-ROM for loading onto the laptop.

Section 2. Comparison Test

Three radiologists were available on the visit to assist in the test. The M65 laptop was taken to the scanning suite and positioned next to the PC running LEONARDO. The radiologists were asked to create a 3D visualization of the MRI data using both the MedX3D standalone, and the LEONARDO software. Only the volume rendering method was used allowing the radiologist to change the opacity settings. Figure 38 shows a typical example of the 3D image produced. Note that as this was a healthy patient, there was no pathology to identify.

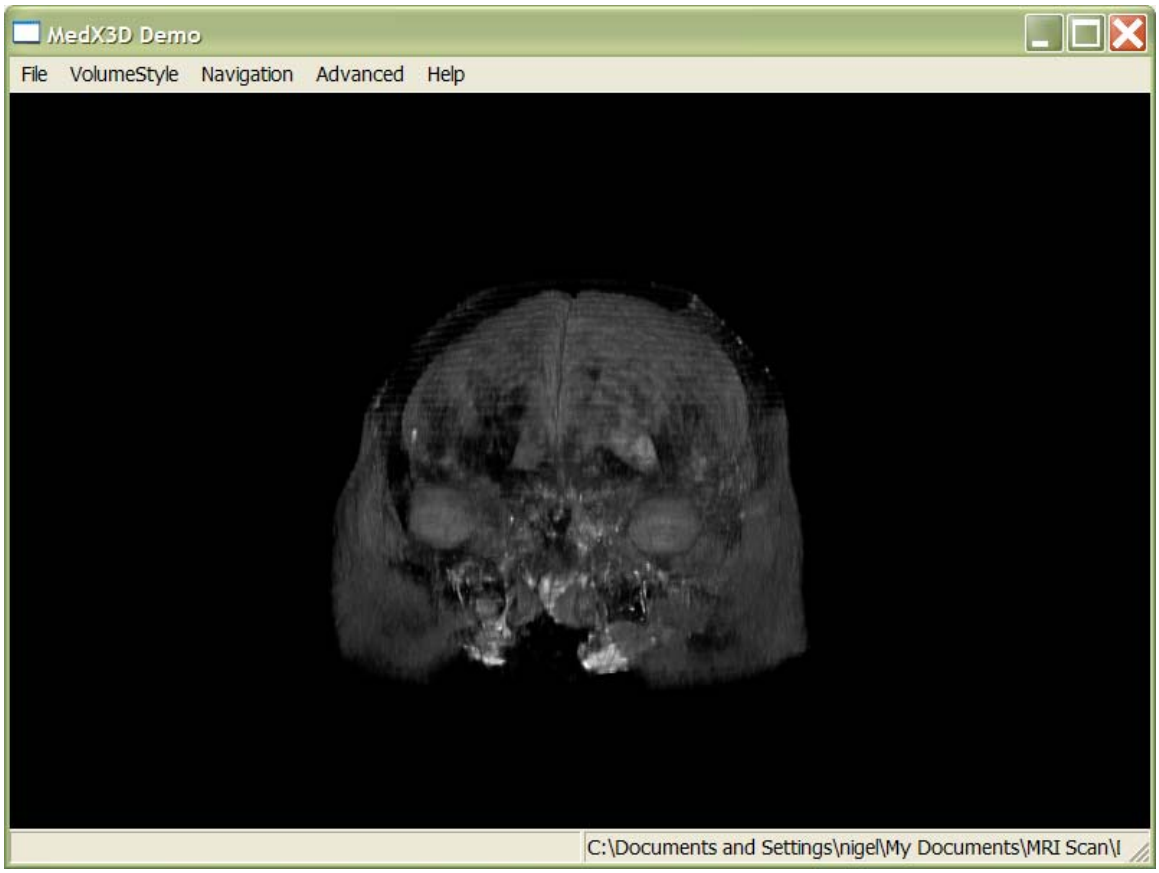


Figure 38: Volume Rendering of MR Head Data in MedX3D Demo software

The radiologists were asked to comment on the following:

- Speed of use: How quickly could they achieve the desired visualization to aid with a diagnosis.
- Quality of rendering: Did MedX3D perform at least as well as the LEONARDO? Was diagnosis easier than using the 2D images?
- Accessibility. How useful would it be to access the 3D imagery from anywhere inside the hospital or even at home?

It would be unfair to compare the functionality of the commercial standard 3D software on the LEONARDO with the relatively simple MedX3D software produced during this project.

Section 3. Results

All three radiologists were able to produce 3D visualizations that they were satisfied with. It took longer to achieve the desired result with the MedX3D software (typically 30-60 seconds) compared with the LEONARDO (typically 20-40 seconds). It was commented that the user interface on the commercial software was superior, and the radiologists often used pre-set values for the look up table (opacity map) that had already been designed to be optimum with different scanning modalities.

The quality of the rendered image achieved was thought to be similar, although it took longer to get to the end point with MedX3D. Opinion on the advantage of 3D over 2D was mixed, however, with one radiologist still preferring to use 2D.

The major disadvantage of the LEONARDO software was its accessibility. There are only three such workstations at the hospital, all located in rooms near to the medical scanners. All radiologists would welcome the same functionality on their office PCs, and were positive about remote access, too. The demonstration of MedX3D running in a web browser was therefore well received. However, deployment would only be possible if security of the data could be guaranteed.

A more extensive study with more participants and covering a wider range of commercial software will be needed to confirm concurrent validation. However, the above results are promising for the future use of MedX3D for 3D medical viewing software.

Key Research Accomplishments

- Creation of an ISO format ready volume rendering component to the open standard, royalty free X3D specification that has been reviewed by 3D graphic experts both inside and outside of the Web3D Consortium
- Creation of a MedicalInterchange profile to the X3D specification that identifies only the parts of the X3D specification necessary for an implementation of a X3D compliant 3D medical image viewer (e.g. volume rendering, annotation, navigation, picking, etc.) for use in medical education, surgical planning and simulation and patient education.
- Creation of an annotation extension to the X3D specification to allow labeling of volumes and objects in a 3D scene using a variety of user interface techniques
- Creation of an open source import/export library in C++ and Visual Basic to encourage and simplify development of 3rd party applications to interface with products adhering to X3D's MedicalInterchange profile
- Creation of open source tools and resources that enable mapping of X3D content to the two most popular anatomical ontologies, FMA and SNOMED
- Creation of a Windows based web browser plugin and standalone application (MedX3D) that can load DICOM files and display 3D volumetric medical images using several different rendering styles by implementing the X3D MedicalInterchange profile and volume rendering component
- Favorable focused comparison of the MedX3D browser to freely available, well established 3D volumetric medical image viewers on various hardware and operating system platforms
- Favorable focused comparison to a commercially available 3D medical image viewing software product
- Creation of an anonymized medical image library for use by Consortium membership for testing 3D medical image display techniques

Reportable Outcomes

As a result of this contract, a paper describing the key accomplishments has been accepted for oral presentation at the Medicine Meets Virtual Reality (MMVR) Conference in January 2008 at Long Beach, CA [9]. Appendix J contains the manuscript as submitted. This research has also stimulated a number of presentations, including those at SIGGRAPH 2007 Birds of a Feather for X3D Medical Working Group, SIGGRAPH 2007 X3D technical showcase, Silicon Valley ACM SIGGRAPH Chapter, ATI and NVIDIA headquarters.

This project has also allowed representatives of the Web3D Consortium to take leadership roles in championing a standard for 3D medical images in the DICOM standards organization, where we have formal membership status. The Consortium has representatives in Working Groups 11 and 17.

At the time of this writing, we have just been made aware of another implementation of the volume rendering extension that is already beyond an early stage of development. This implementation was initiated by Fraunhofer IGD (http://a4www.igd.fraunhofer.de/) and ZGV (http://zgdv.de/) in their Instant Reality X3D browser (http://www.instantreality.org/home/) of their own accord. They are providing valuable feedback for changes and clarifications in the VRC. This spontaneous implementation of the VRC provides validation of its relevance and importance.

Conclusion

The tasks completed for this contract have created potential benefits for medical care along a continuum of end users involved in this market space. At one end of the continuum are the physicians, and the most basic benefit for them is a potential open standard file format for 3D medical images to be transmitted over networks and between applications. Interoperability is a key chokepoint in medical information management systems, so much so that an industry collaboration known as Integrating the Healthcare Enterprise (http://www.himss.org/ASP/topics_ihe.asp) was formed in 1998 with the primary mission of encouraging and promoting standards in healthcare informatics. As the use of 3D medical images becomes more commonplace, with the possibility of 3D eventually replacing 2D images and plain Xrays, a standard file format is paramount to ensure patient safety, clinical efficiency and improvement of patient care. Currently many PACS systems cannot exchange 3D images, and these images take significant resources to process and then manually prepare prior to radiologist interpretation. Not being able to easily exchange these completely processed 3D images between systems creates immediate difficulties for communications between physicians regarding patients' diagnoses and treatments, especially as these images are being relied upon more and more as the primary reference source for medical action. A recent reader poll reported in the February 2007 issue of HealthImaging & IT magazine revealed that 67% of respondents answered no to the question: "Are you able to easily share patient information with other healthcare facilities in your region?"

Physicians also have potential benefit from this work in the area of simulation. The expense in time, money and safety that is part of surgical training is well known [10,11,12], and the advantages of training in virtual environments have been proven, even in low fidelity environments or environments that only train basic psychomotor skills used in surgery such as video games [13]. Simulation would also benefit surgical planning for complex cases and recertification or continuing education scenarios.
By adding volume rendering abilities to the X3D specification, a web based simulator on patient specific data for surgical or procedural training is almost complete, as X3D describes a scenegraph and includes support for a complete visual and aural virtual environment. Admittedly, there are now web based, open source surgical simulators with visual, aural and haptic capabilities becoming available, but they are not based on an open ISO standard data format or scenegraph, and most originate in academic centers. X3D has been adopted by government, academia and industry, is ISO ratified and has been in development for over 10 years, with a focus on integration with web services and infrastructure. Additionally, the API upon which the implementation of the VRC was built also includes haptics capabilities, although these have not been standardized as of yet.

Moving along the continuum, general healthcare providers and ancillary service providers would be assisted by this technology as it would bring 3D medical images to more desktops at a less expensive (or free) price for educational purposes. Again, there are already solutions that provide visualization of 3D medical images (as seen in the comparison tests noted above), but few if any provide support for annotations in 2D or 3D, the infrastructure for custom animations and an open standard file and scenegraph format for the 3D images. Annotations and animations are especially important for communicating complex anatomical relationships and procedures.

This technology would also have significant advantages to the lay person. A web based plugin would bring more access of their own 3D images to the patient, and provide a venue for a physician to improve communication when discussing surgical procedures as dictated by laws for informed consent. Given that many studies show poor retention rates in presurgical education despite concerted efforts [14,15] and that the lack of adequate informed consent is one of the top 10 most common reasons for hospital malpractice claims [16], any adjuncts to physician patient rapport will be extremely useful.

As more and more of our lives become intertwined with the internet, and more business is conducted in virtual space (e.g. Second Life -- http://secondlife.com/), significant collaboration between physician and patient or physician and physician may also take place in like environments. The ability to collaboratively interact with a 3D model in real time is a powerful tool for communication and knowledge transfer. Early versions of X3D had been a part of pioneering multi-user virtual worlds 10 years ago, so it has been built to support this type of activity from an early on. A standard 3D medical image format based on X3D could leverage the X3D specification's infrastructure and accelerate developments in collaboration of 3D medical images virtual worlds.

Besides these trends in collaboration, medical care is about to enter a significant trend where care is more individualized, and imaging studies must necessarily become more microscopic. X3D can easily scale to visualize complex molecular shapes that are paramount in importance for drug discovery and diagnosis. Although there is a standard file format that has been around since the 70's to describe protein structures (PDB -- http://www.wwpdb.org/), there is no corresponding format to additionally encapsulate molecular forces related to bonding, which is important in determining quaternary molecular structure and drug activity. The X3D specification already has the functionality to support this. By establishing itself as a legitimate standard for 3D medical images, X3D becomes a logical choice for standardization of molecular structure and bonding behavior.

Finally, development within the X3D framework for volume rendering leverages the horizontal reach of the X3D specification, providing gains to other vertical market spaces that are not necessarily related to medicine. The oil and gas industry is interested in volume rendering capabilities for visualizing expansive datasets of subterranean spaces in search of fossil fuels. Volume rendering is also important for climatologists to analyze weather prediction and global warming. Scientific visualization depends on volume rendering for computational fluid dynamics. X3D is being used in all of these scenarios already, and standardization of volume rendering functionality will help to improve these visualizations and promote further interoperability of data.

Future work should focus on:

- Improving features and functionality of the MedX3D browser to approach the quality of other freely available volume viewers.
- The MIP and VRC should be revised based on feedback and new rendering techniques. A haptic extension to X3D should be developed to create the final piece for surgical and procedural simulation within an ISO ratified, X3D framework.
- A public image repository should be developed to provide a library of virtual cadavers showing normal anatomy, pathology and normal anatomic variants.
- Conformance testing suites should be developed to provide a mechanism for control of adherence to the X3D standard.
- DICOM participation should be increased
- A second browser implementation utilizing the MIP and VRC should be created (if one has not appeared by then) to allow for entrance of the MIP and VRC into ISO as a formal application for standardization.

References

[1] Megibow, A.J. (2002). Three-D offers workflow gains, new diagnostic options. Diagnostic Imaging. November 2002, 83-93.

[2] F.P. Vidal, F. Bello, K.W. Brodlie, D.A. Gould, N.W. John, R. Phillips, N.J. Avis, "Principles and Applications of Computer Graphics in Medicine", Computer Graphics Forum, Vol. 25 Issue 1, 2006, pp113-137.

[3] Brutzman, D. and Daly, L. (2007). X3D Extensible 3D Graphics for Web Authors. The Morgan Kaufmann Series in Computer Graphics. ISBN 978-0-12-088500-8

[4] Rosse C., Mejino J.V.L. (2003). A reference ontology for biomedical informatics: the Foundational Model of Anatomy. J Biomed Inform. 36:478-500.

[5] Bodenreider, O. and Zhang, S. (2006). Comparing the Representation of Anatomy in the FMA and SNOMED CT. AMIA Annu Symp Proc. 2006; 46–50.

[6] Preim, B and Bartz, D. (2007) Visualization in Medicine. San Francisco: Morgan Kaufman.

[7] Gooch, A., Gooch, B., Shirley, P., and Cohen, E. (1998) A non-photorealistic lighting model for automatic technical illustration. In Proceedings of the 25th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '98. ACM Press, New York, NY, 447-452.

[8] The DICOM Home Page, http:// medical.nema.org/ Last visited November 2007.

[9] John, N.W., Aratow, M., Couch, J., Evestedt, D., Hudson, A.D., Polys, N., Puk, R.F., Ray, A., Victor, K. and Wang, Q. MedX3D: Standards Enabled Desktop Medical 3D (2008) Stud Health Technol Inform. To Appear

[10] Babineau, T.J., Becker, J., Gibbons, G., Sentovich, S., Hess, D., Robertson, S., and Stone, M. The “Cost” of Operative Training for Surgical Residents Arch Surg 2004 Apr; 139(4):366-69.

[11] Farnworth, L.R., Lemay, D.E., Wooldridge, T., Mabrey, J.D., Blaschak, M.J., DeCoster, T.A., Washcer, D.C. and Schenk, R.C. A Comparison of Operative Times in Arthroscopic ACL Reconstruction Between Orthopaedic Faculty and Residents: The Financial Impact of Orthopaedic Surgical Training in the Operating Room Iowa Orthop J. 2001; 21: 31–35.

[12] Gillespie K.N., Campbell C.R. Cost and productivity effects of residency programs in medicine, psychiatry, and surgery. AHSR FHSR Annu Meet Abstr Book. 1995; 12: 26.

[13] Rosser J.C. Jr, Lynch P.J., Cuddihy L., Gentile Rosser J.C. Jr, Lynch P.J., Cuddihy L, Gentile D.A., Klonsky J., Merrell R. Arch Surg. 2007 Feb;142(2):181-6.

[14] Madan A.K., Tichansky D.S. Patients postoperatively forget aspects of preoperative patient education. Obes Surg. 2005 Aug;15(7):1066-9.

[15] Herz D.A., Looman J.E., Lewis S.K. Informed consent: is it a myth? Neurosurgery. 1992 Mar;30(3):453-8.

[16]Glabman M. Top ten hospital malpractice claims [and how to minimize them]. Trustee. 2004;57(2):12-16.

Appendix A

Use Cases Influencing MedX3D Profile Development

Accessibility outside the radiology suite

Until recently, 3D reconstructions of CT, MRI, PET or ultrasound studies were confined to specialized workstations in the radiology department. Thin client server technology has offloaded processing power to centralized servers and allowed home computers to interact with workstation quality volumetric images at workstation quality speeds (if robust bandwidth is available). Additionally, the relentless increase in performance of Graphics Processing Units (GPUs) has enabled volume rendering capabilities and interactive frame rates at an order of magnitude less in price compared to workstations. Still, either solution does require considerable capital investment in hardware and software. An open standard for 3D medical images based on X3D, a standard focused on home computer performance, would bring these images to a wider distribution of physicians, as these products would be free or at a significantly lower cost. Because volumetric images are in more of an intuitive format than cross or sagittal sections, these physicians would also have better comprehension of the image study.

Anatomy Education

Anatomical coursework is difficult in part due to inadequate resources for learning. Cadavers, although arguably the best way to learn anatomy, are always in short supply. Dissection teams composed of several members and limited access are factors that contribute to less time for an individual to explore anatomical relationships and tissue characteristics independently. Textbooks are constrained to 2D presentations, and although software is available with interactive 3D displays, many times the data is illustrative rather than real. Although illustrative visualization is important in education, the ability to visualize volumetric representations of real data, and therefore the ability to visualize pathology as well as normal anatomical variants, is an important part of building an anatomical knowledge base. An open standard would make inexpensive or freely available tools available to students of anatomy to explore real data sets to understand these subtleties without having to have access to an expensive computer or high end server.

Informed Consent

Physicians are required by law to inform patients of the mechanics, risks, benefits and alternatives of invasive procedures and to document this exchange in the form of a signed inform consent form. Frequently physicians find themselves taking significant time to explain more complex procedures and their patients leaving the office without fully understanding the actual procedure. An open, royalty free standard would make inexpensive or free tools available to both physician and patient for understanding and visualizing such procedures. X3D also allows sophisticated simulations so that animations of the actual procedure on the patient's data can be displayed for further clarity.

Custom Prostheses

Matching generic prostheses to a specific patient can be difficult prior to surgery due to projection errors on plain Xrays, unforeseen anatomical limitations or unpredicted complications during surgery. Total joint surgeries require a size inventory of prosthetic products be available as size predictions have nominal accuracy. Specialized prostheses require custom fabrication and therefore would need volume rendered patient specific datasets to compute exact dimensions. An open standard would enable the use of custom prostheses on a wider scale as interoperability problems with different imaging modalities would be limited, allowing confidence in the patient dataset.

Surgical Planning

Complex surgical cases require planning that includes visualization of the approach, tissue exposure and execution of the defined procedure. If thin client server or advanced visualization software is not available to the surgeon, they may be relegated to traditional cross sectional scan views or snapshots of volume rendered images. Using the capabilities of X3D, surgeons can analyze patient volumetric data from their office or home computer to plan their operation, make measurements and label objects. This technology would be relevant to all areas of surgery, especially vascular, orthopedics, head and neck, plastics and cardiothoracic subspecialties. This functionality would also be valuable for dental implant planning and 3D cephalometric analyses for orthodontic and orthognathic surgical planning.

Surgical Education

The surgical modeling and simulation community has until recently been fragmented with numerous surgical simulators, physiologic models and devices available but no significant level of interoperability. No methodology existed to compare features and functionality or to begin to assess surgical skill for the benefits of training and certification. With TATRC encouraging open source releases of simulation engines, patient specific content must also be expressed in a standardized file format to help realize the interoperability vision. Advances in networked haptics and web based surgical simulators make an open standard for 3D medical images based on X3D a natural fit.

Radiation Therapy

Cancer is becoming more prevalent as the population ages. Radiation therapy is an important modality for therapeutic intervention in many forms of cancer. Treatment with this modality is labor intensive and requires significant planning to predict beam reach and path, gantry position and path and beam intersection with pathologic tissue. Multiple applications need to be accessed to complete this planning but some of the questions aren't even answered (prediction of collision of gantry with patient, bed, floor). This can lead to slower turnaround times with an always overbooked radiation therapy device. Using X3D as a standard for 3D medical images, it can provide an integrated solution and complete preplanning package using patient and machine specific data to predict collisions and analyze beam penetration customized for the individual.

Appendix E

Methodology for Interfacing DICOM files with X3D

This description will not be limited to a particular programming language. Instead, it will provide guidance to programmers on how they can use this type of data in their own X3D-based applications.

DICOM Overview

Digital Imaging and Communications in Medicine (DICOM) is a standard for handling, storing, printing, and transmitting information in medical imaging. It includes a file format definition and a network communications protocol. The communication protocol is an application protocol that uses TCP/IP to communicate between systems. DICOM files can be exchanged between two entities that are capable of receiving image and patient data in DICOM format. The National Electrical Manufacturers Association (NEMA) holds the copyright to this standard [1]. It was developed by the DICOM Standards Committee, whose members [2] are also partly members of NEMA [3].

DICOM enables the integration of scanners, servers, workstations, printers, and network hardware from multiple manufacturers into a Picture Archiving and Communication System (PACS). The different devices come with DICOM conformance statements which clearly state the DICOM classes they support. DICOM has been widely adopted by hospitals and is making inroads in smaller applications like dentists' and doctors' offices. See reference [1] for a good introduction to DICOM.

DICOM Terms

IOD - Information Object Definitions
Information objects define the core contents of medical imaging such as images, reports, and patients, whose function is to carry information; entities in an E-R model whose descriptive attributes have been listed and defined.

Service Class - A set of functions performed to communicate between layers within a device

SOP classes - Service-object pair

SOP instance - A patient specific instance of imagery and patient information

Message - Communication version of the SOP class that provides the specified service and the data set made up of the properly encoded information object instance.

See reference [2] for a good technical description of DICOM.

DICOM Storage Methodology

The DICOM standard stores information in a tag value encoded pair. The tag is a unique ID composed of 2 integers that describes the name of an attribute. The attribute describes some feature of the image. Most DICOM libraries provide a mechanism for easily accessing the tag's value in a DICOM file.

DICOM Aspects Most Relevant to X3D Implementations

There are two features of DICOM that are most relevant to X3D implementations incorporating DICOM images. These parts deal with storage of images on a local file system and access to these images from a web service.

3.10 - Media Storage and File Format for Data Interchange
Part 10 - offline file format. See reference (3) for file level details.

3.18 - Web Access to DICOM Persistent Objects (WADO)
Part 18 – Web access. See reference (4) for transmission details.

Toolkits Implementing DICOM

The most likely route for an X3D implementation to use DICOM is to use a toolkit to read the files. The following toolkits for C and Java might be useful.

C/C++:
DCMTK - DICOM Toolkit http://dicom.offis.de/dcmtk.php.en
David Clunie's Library: http://www.dclunie.com/dicom3tools.html

Java:
dcm4che - DICOM implementation in Java: http://sourceforge.net/projects/dcm4che/
JDDK(Java DICOM Development Kit)
ImageJ: http://rsb.info.nih.gov/ij/

Ultimate resource for all things DICOM: http://www.idoimaging.com/index.shtml

References

1 - http://en.wikipedia.org/wiki/Digital_Imaging_and_Communications_in_Medicine
2 - http://www.rsna.org/Technology/DICOM/intro/elemental.cfm
3- http://medical.nema.org/dicom/2007/07_10pu.pdf
4 - http://medical.nema.org/dicom/2007/07_18pu.pdf

Appendix G

Mapping Between X3D and SNOMED/FMA

Section A: *contents of the tarball*

http://snoid.sv.vt.edu/~anray2/x3dMetadata.tar.gz

```
x3dMetadata
      finalReport.doc
      fmaHtmlParser
            header.xml
            README.txt
            script1
            Serosa+of+right+hemiliver.html
            Upper+lobe+of+lung.html
            parse.cpp
            script0
            script2
            Serosa_of_right_hemiliver.xml
            Upper_lobe_of_lung.xml
      html_out.tar.bz2
      index.html
      README.txt
      xslt
            fmaOut.xml
            FMA.xml
            FMA.xslt
            README.txt
            sct_xml_cps_sample.xml
            snoModified.xml
            snoOut.xml
            SNO.xslt
```

Section B: *FMA Setup*

This section deals with where to go to obtain the resources necessary to transform FMA information into X3D Metadata. It goes over how to obtain the database, how to import the database into a MySQL database, how to install Protege (ontology program that talks with the database), how to get usable data out of Protege, and how to transform this data into X3D Metadata.

> To obtain the FMA database, you must agree to the license found here:
> http://sig.biostr.washington.edu/projects/fma/license_faq.html

Once this is done you have a somewhat involved process for getting the FMA to work. The first part is obtaining the database, next is loading the database into MySQL. Next is downloading Protege, installing the Protege extensions, and configuring Protege to talk to MySQL. Lastly, you have to export the FMA project from Protege into the html output form.

FMA database:
To obtain dump file: “Please use the username "opensource" and the password "fma2007" when downloading the FMA dump file. We apologize for the inconvenient download procedure, but we are

currently investigating better methods."
http://fma.biostr.washington.edu/latest_version

This is simply a file which holds the FMA in a format that MySQL can handle. Once you have downloaded the file you can install it into MySQL using the instructions on this website:
http://sig.biostr.washington.edu/projects/fm/FMA_Release/FMA_instructions.html

An example FMA SQLrecord:

```
<FMA>
        <frame>26212</frame>
        <frame_type>6</frame_type>
        <slot>2002</slot>
        <facet>0</facet>
        <is_template>0</is_template>
        <value_index>0</value_index>
        <value_type>3</value_type>
        <short_value>Liver</short_value>
</FMA>
```

An example of how this can be transformed into target X3D metadata is as follows:

```
<X3D profile="Immersive" version="3.1" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-3.1.xsd">
<head>
        <meta content="X3D metadata description of a FMA mysql xml document." name="description"/>
        <meta content="Written by a xslt template created by Andrew Ray." name="creator"/>
        <meta content="Program created on 9/23/2007." name="created"/>
        <meta content="Copyright (c) of generating author" name="rights"/>
        <meta content="FMA, SNOMED." name="subject"/>
</head><Scene>
<MetadataSet DEF="m123456" name="FMA-XML" reference="FMA">
        <MetadataInteger name="frame" value="26212"/>
        <MetadataInteger name="frame_type" value="6"/>
        <MetadataInteger name="slot" value="2002"/>
        <MetadataInteger name="facet" value="0"/>
        <MetadataInteger name="is_template" value="0"/>
        <MetadataInteger name="value_index" value="0"/>
        <MetadataInteger name="value_type" value="3"/>
        <MetadataString name="short_value" value="Liver"/>
</MetadataSet>
</Scene>
</X3D>
```

Once the database is installed and working and you want to produce X3D metadata in HTML output, then you need to install a program called Protege that knows the format for the database (it isn't setup the way you think it is). This program can then output the FMA in a saner format for processing.

Protege Instructions:
http://sig.biostr.washington.edu/projects/fm/FMA_Release/setup-protege.html

If these instructions do not work please go to this URL and follow the instructions that the FMA developers provide:
http://depts.washington.edu/ventures/UW_Technology/Express_Licenses/FMA.php

As a last resort, here is the FMA homepage, find out how to obtain it from there:
http://sig.biostr.washington.edu/projects/fm/FAQs.html

Generating HTML from Protege:
Once all of this working, the steps to producing X3D metadata are rather simple. The first is start Protege, load the FMA project file, click file, click 'export to format', then click 'HTML'. Specify the directory for the html to be generated and then wait. This process can literally take several days. This process will generate approximately 1.1gb of data.

A bzip'd tarball that contains the HTML generated from Protege if you wish to save yourself the work of generating it can be found in the release tarball or at the following website. This is a 12 MB download that expands to 1.1 GB.
http://snoid.sv.vt.edu/~anray2/html_out.tar.bz2

Transforming HTML into X3D Metadata:
The requirements for this program are a linux system that can compile C++ code, and can run sed on script files. The parser can found in the release tarball in the directory **fmaHtmlParser** or be downloaded at:
http://snoid.sv.vt.edu/~anray2/parser.tar.gz .

Building the parser can be done via typing "make parse" inside of the parser directory. There are two example html files in this directory, and two xml files that show the output of the parser.

The program runs by giving it a single html file to convert. This program will then convert the html file into an xml file named by a sanitized version of the html title (sanitized to only contain the FMA name delimited by _'s not spaces).

Example:
./parse Liver.html
Produces:
Liver.xml

An example of the metadata produced by this program:

```
<X3D profile="Immersive" version="3.1" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-3.1.xsd">
<Scene>
<Transform DEF="Body">
<Group>
<MetadataSet DEF="m7334" name=" Upper_lobe_of_lung" reference="FMA">
        <MetadataString name="has inherent 3-D shape" value="true"/>
        <MetadataSet name="regional part" reference="FMA">
                <MetadataString name="SetName" reference="FMA" value="Bronchopulmonary_segment"/>
                <MetadataString name="SetName" reference="FMA" value="Bronchopulmonary_segment"/>
                <MetadataString name="SetName" reference="FMA" value="Bronchopulmonary_segment"/>
        </MetadataSet>
        <MetadataString name="regional part of" value="blank"/>
        <MetadataSet name="constitutional part" reference="FMA">
                <MetadataString name="SetName" reference="FMA" value="Parenchyma_of_lobe_of_lung"/>
                <MetadataString name="SetName" reference="FMA" value="Lobar_bronchial_tree"/>
                <MetadataString name="SetName" reference="FMA" value="Pleura_of_lobe_of_lung"/>
                <MetadataString name="SetName" reference="FMA" value="Neural_network_of_lobe_of_lung"/>
```

```
                <MetadataString name="SetName" reference="FMA" value="Vasculature_of_lobe_of_lung"/>
        </MetadataSet>
        <MetadataString name="dimension" value="3-dimension"/>
        <MetadataString name="has dimension" value="true"/>
        <MetadataString name="has boundary" value="true"/>
        <MetadataInteger name="FMAID" value="7334"/>
        <MetadataString name="Preferred name" value="Upper_lobe_of_lung"/>
        <MetadataSet name="Synonym" reference="FMA">
                <MetadataString name="SetName" reference="FMA" value="Lobus_superior"/>
                <MetadataString name="SetName" reference="FMA" value="Superior_lobe_of_lung"/>
        </MetadataSet>
        <MetadataSet name="part" reference="FMA">
                <MetadataString name="SetName" reference="FMA" value="Parenchyma_of_lobe_of_lung"/>
                <MetadataString name="SetName" reference="FMA" value="Lobar_bronchial_tree"/>
                <MetadataString name="SetName" reference="FMA" value="Pleura_of_lobe_of_lung"/>
                <MetadataString name="SetName" reference="FMA" value="Bronchopulmonary_segment"/>
                <MetadataString name="SetName" reference="FMA" value="Bronchopulmonary_segment"/>
                <MetadataString name="SetName" reference="FMA" value="Bronchopulmonary_segment"/>
        </MetadataSet>
        <MetadataString name="part of" value="Lobular_organ"/>
        <MetadataString name="Non-English equivalent" value="Lobus_superior_pulmonis"/>
        <MetadataString name="constitutional part of" value="blank"/>
        </MetadataSet>
</Group>
</Transform>
</Scene>
</X3D>
```

Section C: *Converting SNOMED to X3Dmetadata*

The first step that has to be done is to install a XSLT processor. The one used when creating this program can be found here: http://xmlsoft.org/XSLT/xsltproc2.html. Once the processor is installed and working then the next step is to download the SNOMED to X3D metadata XSLT file. The file can be found at http://snoid.sv.vt.edu/~anray2/xslt/SNO.xslt. The next step is to strip the following lines out of your XML file:

```
<snomedCt xmlns="urn:snomed-org/sct" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:snomed-org/sctSchema\SnomedCt.xsd">
</snomedCt>
```

This is due to the limitations of xsltproc, it cannot parse this tag. The next necessary addition is to have the xml file begin with <test> and end with </test>. Once this is done then you can transform SNOMED xml files to X3D metadata files by typing:

xsltproc SNO.xslt yourxmlfilehere.xml

An example of SNOMED input:

```
<concepts>
        <concept conceptId="369445005" conceptStatus="0" fullySpecifiedName="Chronic proctocolitis (disorder)"
                ctv3id="XUU7a" snomedId="D5-45285" isPrimitive="1">
                <descriptions>
                        <description descriptionId=" 774149015  term="Chronic proctocolitis"
                                descriptionType="3" initialCapitalStatus="0" languageCode="en"
                                descriptionStatus="0">
```

```
                        <history releaseVersion="20020131" changeType="0" reason="null"/>
                    </description>
                </descriptions>
                <relationshipSet>
                    <relationship relationshipId="2398318025" relationshipType="24611200006"
                        conceptId2="3853150009" characteristicType="1" refinability="1"/>
                    <relationshipGroup>
                        <relationship relationshipId="1082140022"
                            relationshipType="116676008" conceptId2="23583003"
                            characteristicType="0" refinability="2"/>
                        <relationship relationshipId="1082141021" relationshipType="363698007"
                            conceptId2="34402009" characteristicType="0" refinability="2"/>
                    </relationshipGroup>
                </relationshipSet>
        </concept>
</concepts>
```

Although there are several integer examples in this example, all of them are represented as strings in our transformation because if the value is not valid, it may be set to “null.” An example of the transformed XML into X3D metadata is:

```
<MetadataSet DEF="m123456" name="SNOMED-XML" reference="SNOMED">
<MetadataSet name="concepts">
        <MetadataSet name="Concept " reference="SNOMED">
                <MetadataString name="conceptId" value="369445005"/>
                <MetadataString name="conceptStatus" value="0"/>
                <MetadataString name="fullySpecfiedName" value="Chronic proctocolitis (disorder)"/>
                <MetadataString name="ctv3id" value="XUU7a"/>
                <MetadataString name="snomedId" value="D5-45285"/>
                <MetadataString name="isPrimitive" value="1"/>
                <MetadataSet name="Description" reference="SNOMED">
                        <MetadataString name="descriptionId" value="774149015"/>
                        <MetadataString name="term" value="Chronic proctocolitis (disorder)"/>
                        <MetadataString name="descriptionType" value="3"/>
                        <MetadataString name="initialCapitalStatus" value="0"/>
                        <MetadataString name="languageCode" value="en"/>
                        <MetadataString name="descriptionStatus" value="0"/>
                </MetadataSet>
                <MetadataSet name="History" reference="SNOMED">
                        <MetadataString name="releaseVersion" value="20020131"/>
                        <MetadataString name="changeType" value="0"/>
                        <MetadataString name="reason" value="null"/>
                </MetadataSet>
                <MetadataSet name="relationshipSet">
                        <MetadataSet name="relationship" reference="SNOMED">
                                <MetadataString name="relationshipId" value="2398318025"/>
                                <MetadataString name="relationshipType" value="246100006"/>
                                <MetadataString name="conceptId2" value="385315009"/>
                                <MetadataString name="characteristicType" value="1"/>
                                <MetadataString name="refinability" value="1"/>
                        </MetadataSet>
                        <MetadataSet name="relationship" reference="SNOMED">
                                <MetadataString name="relationshipId" value="2398319022"/>
                                <MetadataString name="relationshipType" value="246100006"/>
                                <MetadataString name="conceptId2" value="61751001"/>
                                <MetadataString name="characteristicType" value="1"/>
                                <MetadataString name="refinability" value="1"/>
                        </MetadataSet>
                        <MetadataSet name="relationshipGroup">
```

```
                        <MetadataSet name="relationship" reference="SNOMED">
                                <MetadataString name="relationshipId" value="1082140022"/>
                                <MetadataString name="relationshipType" value="116676008"/>
                                <MetadataString name="conceptId2" value="23583003"/>
                                <MetadataString name="characteristicType" value="0"/>
                                <MetadataString name="refinability" value="2"/>
                        </MetadataSet>
                        <MetadataSet name="relationship" reference="SNOMED">
                                <MetadataString name="relationshipId" value="1082141021"/>
                                <MetadataString name="relationshipType" value="363698007"/>
                                <MetadataString name="conceptId2" value="34402009"/>
                                <MetadataString name="characteristicType" value="0"/>
                                <MetadataString name="refinability" value="2"/>
                        </MetadataSet>
                    <!--End relationshipGroup tag -->
                    </MetadataSet>
                <!--End relationshipSet tag -->
                </MetadataSet>
        <!--End concept tag -->
        </MetadataSet>
<!--End concepts tag -->
</MetadataSet>
<!--Final closing tag -->
</MetadataSet>
```

Appendix H

Import/Export Library

1) Installation and Redistribution

Before the SDK can be used, it must be installed. To install the Flux Import Export SDK, run the SetupFluxIOSDK.exe program. It will install the SDK. In addition to placing several files on your harddrive, it also registers the FluxIO.dll, which registers the COM interfaces.

The SDK installer will place the following folders in the installation folder:

The "include" folder contains a couple of headers files that your application will need to include.
The "bin" folder contains the dlls.
The "cpp_testcase" folder contains a test harness written in C++.
The "vb_testcase" folder contains a test harness written in Visual Basic.

If your application uses this library, you will need to redistribute the library with your installation. To do so, please redistribute the dlls that are found in the "bin" folder of the SDK installation. Your installer must also register the FluxIO.dll file. One way to do so is to use the win32 command: regsvr32 FluxIO.dll

The installation of the Flux Import Export SDK will not interfere with the installation of Flux Player, Flux Studio, or any other X3D product. No files types are associated.

2) Getting Started with the SDK

This section is a brief tutorial on how to quickly get up and running with the SDK using C++. It assumes little knowledge of the X3D SAI. It does assume some minimal knowledge of COM. Readers who are familiar with the SAI and COM might find this section boring.

We suggest that you open up the sample testHarness application, and MSVC V6 project. This shows all the code that is discussed. Note, the code listed here is somewhat simplified.

1) With any COM application, you must first initialize the COM system, with a call to:

```
HRESULT  hr = CoInitialize(0);
```

Make sure you have a matching call to CoUninitialize(); when you are finished using COM.

Also, you must include the include files for the SDK (you will need to check the paths) :

```
#include "..\FluxIO\FluxIO.h"
#include "..\FluxIO\FluxIO_i.c"
```

2) The first step in getting at the interfaces supplied in this API is to instantiate an X3Dbrowser object. To do so, use the standard COM call to instantiate a COM object:

```
X3DBrowser                          *pIX3DBrowser = NULL;
hr = CoCreateInstance( CLSID_CoX3DBrowser, NULL, CLSCTX_INPROC_SERVER,
                       IID_X3DBrowser, (void**) &pIX3DBrowser);
```

As is the case with all calls to COM methods, the application should check the return code before proceeding.

3) Once you have an empty Browser Object, you should probably fill it with content. The quickest way to do this is to read the content from a file. This is a two step process; first read the url, which returns a pointer to a Scene, and call replaceWorlds with our new Scene Node:

```
X3DScene                            *pIX3DScene = NULL;
hr = pIX3DBrowser->createX3DFromURL(_bstr_t( PathIn ), &pIX3DScene );

if(SUCCEEDED(hr)){
  pIX3DScene->AddRef();
  hr = pIX3DBrowser->replaceWorld( pIX3DScene );
```

4) Now that we have a Scene, lets try to get a pointer to a Node using the Node's DEF name:

```
X3DNode                 *pIX3DNode = NULL;
hr = pIX3DScene ->getNode( _bstr_t( TargetNodeName ), &pIX3DNode );
```

5) If we found a Node, we can now get a field on that Node. Using the SAI, you can not directly manipulate a Node. Instead, you access the Fields on the Node. They contain the attributes of the Node. You can then read and write the values of those fields using the SAI.
Let's try to get a field via its name. There are also methods that let you traverse through the fields associated with a particular node. The following line gets us a pointer to the Field:

```
X3DField* pIX3DField;
hr = pIX3DNode->getField( _bstr_t(TargetFieldName ), &pIX3DField );
```

Note: we have received the base Field class that is independent of the type of value contained in the field. Before we can do anything significant with that field, we must safely get an interface to the specific type of Field. It is the specific interface that lets us read and write the field values.

6) We need to cast it to the correct type of field. This sample code is specifically looking for a SFFloat field.

```
X3DFieldType fieldtype;
pIX3DField->getType( &fieldtype );

if( fieldtype == X3DSFFloatType ) {
  X3DSFFloat* pIHeightField = ( X3DSFFloat* ) pIX3DField;
```

7) If we have a field, first lets take a peek at the previous value of the field:

```
float val0 = 0.0;
pIHeightField->getValue( &val0 );
```

8) Next, lets set it to a new value:

```
val0 = NewFloatValue;
pIHeightField->setValue( val0 );
```

9) We could continue on, adding and removing Nodes, and modifying field values. But, at this point, we will just export the modified content to a file.

```
hr = pIX3DBrowser->export(_bstr_t(PathOut ) );
```

10) At any time, we might want to look in the Console window to see any warnings or errors.
Since we don't have a console Window, the API provides access to the text in the Console. This feature was added to the API to give application developers some feedback when files do not load.
You can Pop the string off of the top of the console message list:

```
CString NewConsoleText;
BSTR bsLine;
while( SUCCEEDED( pIX3DBrowser->popConsoleLine( &bsLine ) ) ) {
  char ConsoleLine[410];
  wcstombs(ConsoleLine, bsLine, 400);
  NewConsoleText += ConsoleLine + LineBreak;
}
```

Those simple steps should get you started using the Flux Import Export SDK. For a detailed description of all of the methods in the SDK, please see the SDK reference doc.

3) API Reference guide:

Since the specification listed above provides an abstract definition of the interface, this document will explicitly declare the COM interfaces. This section describes those interfaces using C++.

Again, this API is a subset of the X3D SAI, with many methods removed due to the lack of runtime support in the Library. In addition, several methods have been added to satisfy several requirements of this project. They are listed here:

X3Dbrowser::export
X3Dbrowser:: popConsoleLine
X3Dbrowser:: setDocTypeHeader
X3Dbrowser:: getDocTypeHeader

The following document has been taken from the specification listed above, and the declaration of the explicit interfaces have been added.

6.3 Browser services

6.3.1 Introduction

The following services can be requested from a browser. Although not specified, all services are capable of throwing an SAI_CONNECTION_ERROR whenever a request is made if the session between the application and the browser has failed. *Note:* The data representation of the parameters or return values are not specified. It could be equally valid to represent all parameters as strings as it is for binary representations.

6.3.2 getName

HRESULT getName(/* [retval][out] */ BSTR *name);
Return Values: S_OK, E_FAIL
Parameters:
BSTR* name; The return unicode value containing the name.

The `getName` service returns the name of the browser. This name is implementation dependent. If this service is not supported a `NULL` value shall be returned.

6.3.3 getVersion

HRESULT getVersion(/* [retval][out] */ BSTR *name);
Return Values: S_OK, E_FAIL
Parameters:
BSTR* name; The return unicode value containing the version string of the browser.

The `getVersion` service returns the current version of the browser application. The version number of the browser is implementation dependent. If this service is not supported then a `NULL` value shall be returned.

6.3.10 getExecutionContext

HRESULT getExecutionContext (/* [retval][out] */ X3DExecutionContext **ppContext);
Return Values: S_OK, E_FAIL
Parameters:
(/* [retval][out] */ X3DExecutionContext **ppContext. The pointer to the requested interface.

The `getExecutionContext` service returns the current execution context. If used in an internal interaction, this service returns the execution context in which the containing node exists (see 4.4.3 Containing Node). When used in an external interaction, this service returns the current top-level scene.
The execution context is the base form of a scene, but only provides access to the local nodes, PROTOs and routes as defined by the X3D name scoping rules as defined in *4.4.7 Run-time name scope* in Part 1 of ISO/IEC 19775 (see 2.[I19775-1]). Depending on the place in the scene graph, the returned type may be an instance of SAIScene allowing the user to use the greater capabilities.

6.3.12 replaceWorld

HRESULT replaceWorld (/* [in] */ X3DScene *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ X3DScene *value. A pointer to the X3Dscene that you with to insert into the Browser. Typically, such an X3Dscene can be obtained by a call to createX3DFromURL or CreateX3DfromString.

The `replaceWorld` service replaces the current world with the world specified by the SAIScene parameter. If another `replaceWorld` or `loadURL` (see 6.3.14 loadURL) request is made during the processing of the current request, the current request is terminated and the new one started. In this case, no extra shutdown event shall be generated. The initialize event shall be generated at the point where the world is ready to be run. The scene is not required to contain any valid content. Setting a value of `NULL` shall clear the currently set scene and leave a blank browser with no renderable content and no current scene.
The SAI_Browser_Shutdown event is generated immediately upon receiving this service request.
The SAI_Browser_Initialized event is generated when the new nodes have been set and all browser specific initialization has taken place but before the first time driven event cascade has been started (event cascades may have previously resulted due to the initialization process through internal scripts).

6.3.14 loadURL

HRESULT loadURL (
/* [in] */ int nUrls,
/* [size_is][in] */ BSTR *url,
/* [in] */ int nParams,
/* [size_is][in] */ BSTR *params);

Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int nUrls. Number of URLs in the url array.
/* [size_is][in] */ BSTR *url, An array of pointers to Unicode strings containing the location of the content to load.
/* [in] */ int nParams. Number of parameters in the params array.
/* [size_is][in] */ BSTR *params. Array of command line parameters to be passed into the Browser.
Note: This is an asyncronisis call to load a scene. It is strongly advised that you use the syncronis createX3DFromURL method.

The `loadURL` service inserts the content identified by the URL(s) in the current world under control of the contents of the SAIPropertyList instance.
The SAI_Browser_Shutdown event is generated immediately upon receiving this service request if the parameter list is such that the browser is about to be shutdown (EXAMPLE replaces an HTML Frame where the browser was embedded).
The SAI_Browser_Initialized event is generated when the new nodes have been set and all browser specific initialization has taken place but before the first time driven event cascade has been started (event cascades may have previously resulted due to the initialization process through internal scripts).
The property list definition shall include at least one property that defines loading the URL supplied as a new world in the supplied SAIBrowserRef. If the property list is empty, the action is to replace the world of the current browser with the new world if the successful URL is a X3D file.
If another `replaceWorld` (see 6.3.12 replaceWorld) or `loadURL` request is made during the processing of the current request, the current request is terminated and the new one started. In this case, no extra shutdown event shall be generated. The SAI_Browser_Initialized event shall be generated at the point where the world is ready to be run if `replaceWorld` was called.

6.3.15 setDescription

HRESULT setDescription (
/* [in] */ BSTR description);

Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR description. The descriptioon of the Scene.

If the browser supports a description title, it shall be set to the new description. Typically, this will be the title in a window title bar. In cases where there may be multiple browsers on a single window, the result is implementation dependent.

6.3.16 createX3DfromString

HRESULT createX3DFromString (
/* [in] */ BSTR x3dSource,
/* [retval][out] */ X3DScene **scenereturn);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR x3dSource. A unicode string that contains the raw X3D content.
/* [retval][out] */ X3DScene **scenereturn), A pointer to a pointer. Apon a sucsesful return, it will point to the new X3DScene that is generated from the string provided.

The `createX3DFromString` service creates nodes from a string. The string shall contain valid X3D syntax; otherwise an error is generated. If any relative URLs are encountered in this string, the base is assumed to be the current browser location. The string is not required to contain the X3D file header. If it is present, it shall be treated as an indicator to the version of X3D contained. If absent, the default version assumed shall be that specified in *7.2.4.2 Header statement* in part 1 of ISO/IEC 19775 (see 2.[I19775_1]). A browser is not required to support any versions prior to ISO/IEC 19775. If the string contains legal X3D statements but does not contain any node instances, a valid SAIScene value shall still be returned containing no root nodes, but with the appropriate declaration identifiers. For example the string may contain EXTERNPROTO declarations but no instances of any node. If the SAIString provides the content in an encoding format that the browser implementation does not support, the browser shall generate an SAI_NOT_SUPPORTED error.

6.3.18 createX3DFromURL

HRESULT createX3DFromURL (
/* [in] */ BSTR url,
/* [retval][out] */ X3DScene **scenereturn);
Return Values: S_OK, E_FAIL
Parameters:

/* [in] */ BSTR url. A unicode url with the location of the X3D content.
/* [retval][out] */ X3DScene **scenereturn), A pointer to a pointer. Apon a sucsesful return, it will point to the new X3DScene that is generated from the url provided.

The `createX3DFromURL` service creates nodes from the contents of the file represented by the URL. The URL may be a relative URL which is considered to be using the browser location as the base document. The scene described by that URL shall be identified by the returned SAIScene value. the events defined in 4.5.3 Browser to External Application.

6.3.20 addBrowserListener

HRESULT addBrowserListener (
    /* [in] */ IDispatch *pListener);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ IDispatch *pListener);

The `addBrowserListener` service requests that the listener will receive Browser callbacks from the Browser.

6.3.21 removeBrowserListener

HRESULT removeBrowserListener (
    /* [in] */ IDispatch *pListener);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ IDispatch *pListener);

The `removeBrowserListener` service requests that the specified listner should be removed from the list of Browser Listeners.

6.3.24 print

HRESULT print (
    /* [in] */ BSTR str);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR str. The string that will be appended to the console text. The console string buffer can be read via the Browser's method popConsoleLine.

The `print` service prints a message to the browser's console. The language-specific bindings may provide overloaded variations on this service that do not take an SAIString value, but take other data types. Other variants may include the

ability to automatically add linefeed/newline characters without the need to explicitly declare them in the SAIString value. A binding shall provide at least the base SAIString variant.
User code may call this service at any time, without restriction, unless the browser reference has been disposed of.

6.3.25 dispose

HRESULT print (void);
Return Values: S_OK, E_FAIL
Parameters: none

The `dispose` service indicates that the client is about to exit this session and the browser is free to dispose of any resources that this client may have consumed. An SAI_Browser_Shutdown event is sent only to this client and may be generated internally by the language implementation on the client machine (that is, it is not required that the browser itself generate this event, just that the event is generated). If any events have been held because `BeginUpdate` has been called, disposing of the browser shall also call `EndUpdate` to release those events to the browser.

6.3.26 export

HRESULT export (/* [in] */ BSTR outputFileName);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR outputFileName

The `export` service will export the current contens of the main scene to the file location specified by the outputFileName parameter.

6.3.27 setDocTypeHeader

HRESULT setDocTypeHeader (/* [in] */ BSTR outputFileName);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR value

The setDocTypeHeader will set the XML document header to the string specified. This is the header sting that will be used when the scene is exported. It is typically the sceond line in the X3D content file. It oftem looks like this:
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">

6.3.28 getDocTypeHeader

HRESULT getDocTypeHeader(/* [retval][out] */ BSTR *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR *value

The getDocTypeHeader will get the XML document header. This is the header sting that will be used when the scene is exported. It is typically the sceond line in the X3D content file. It oftem looks like this:
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">

6.3.29 popConsoleLine

HRESULT popConsoleLine /* [retval][out] */ BSTR *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR *value

The popConsoleLine service will pop the top line off of the buffer of text strings that have been written tote console. This information is important if the content should fail to load.

6.4 Execution context services

6.4.1 getSpecificationVersion

HRESULT getSpecificationVersion(
    /* [retval][out] */ BSTR *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR *value. The uncode value that will contain the Specification Version.

The getSpecificationVersion returns the version string that describes to which specification version this scene adheres. This version represents the appropriate version number as defined in part 1 of ISO/IEC 14772 (see 2.[I14772-1]), part 1 of ISO/IEC 19775 (see 2.[I19775-1]), or has value 1.0 for versions of VRML that precede the specification in part 1 of ISO/IEC 14772-1 that are supported by the implementation.

6.4.2 getEncoding

HRESULT getEncoding (
    /* [retval][out] */ X3DEncodingType *encodingType);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DEncodingType *encodingType. The returned value of the Encoding type.

The `getEncoding` service returns the encoding type that was used for to produce the portion of the scene represented by the specified execution context. The encoding is one of a fixed set, but may include additional types that are browser implementation specific. The minimum required set of values (but not necessarily supported by the browser implementation) shall be:

1. *Scripted*: For scenes that are created completely through the SAI and did not originate through a file somewhere.
2. *ASCII*: For VRML 1.0 specification files.

3. *VRML*: For VRML and the X3D Classic VRML encoding (see 2.[I19776-2]).
4. *XML*: For X3D XML-encoded files (see 2.[I19776-1]).
5. *Binary*: for X3D Binary-encoded files (reserved for future specification)
6. *BIFS*: For MPEG-4 BIFS-encoded format. (see 2.[I14496-1])

6.4.6 getNode

HRESULT getNode (
/* [in] */ BSTR name,
/* [retval][out] */ X3DNode **value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR name. A unicode string that has the DEF name fo the target Node.
/* [retval][out] */ X3DNode **value. The returned pointer to the Node that was found, based on the DEF name.

The `getNode` service searches for a node based on specified criteria and returns an identifier for the node.
The SAIString is to identify the name of the node that has been marked with one of the naming statements DEF, IMPORT or EXPORT in the currently loaded X3D scene or previously added with an `namedNodeHandling` request (see 6.4.9 namedNodeHandling).
The SAIAction shall indicate which of the naming types shall be used to find the node. For example, providing an action of `ImportNode` shall not return a name that may be valid, but describes a node named with the DEF statement. Table 6.8 defines the actions specified in this part of ISO/IEC 19775.

Table 6.8 — getNode SAIAction values

Service	Action Type
getNode	DEFNode
	IMPORTNode

	EXPORTNode

Access shall only be available to names in this scene. DEFs in Inlined files shall not be accessible in accordance with *4.4.3 DEF/USE Semantics* and *4.4.6, Import/Export semantics* in part 1 of ISO/IEC 19775 (see 2.[I19775-1]).
If the SAIAction is IMPORTNode and the name is valid but the node definition is not yet available from the source Inline node, SAI_NODE_NOT_AVAILABLE shall be generated.

6.4.7 createNode

HRESULT createNode (
/* [in] */ BSTR type,
/* [retval][out] */ X3DNode **value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR type. A unicode string that has Type of Node to eb created.
/* [retval][out] */ X3DNode **value. The returned pointer to the Node that was created.

The createNode service creates a new default instance of the node given by the SAIString value containing the name of an X3D node type. The availability of the node is defined by the containing scene's profile and component declarations. The name shall only refer to a built-in node and shall not be used to create instances of PROTOs or EXTERNPROTOs. If the node is not available in the current profile and component spaces, the browser shall generated the SAI_INVALID_NAME error.

6.4.10 getProtoDeclaration

HRESULT getProtoDeclarations (
/* [retval][out] */ X3DProtoDeclarationArray **protodeclarations);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DProtoDeclarationArray **protodeclarations. The returned array of Proto Declararations in this context.

The getProtoDeclaration service returns the named PROTO declaration representation from this scene. This shall only be used to request PROTO declarations. A request for an EXTERNPROTO declaration shall generate SAI_INVALID_NAME.

6.4.14 getRootNodes

HRESULT getRootNodes (
/* [retval][out] */ X3DNodeArray **rootnodes) = 0;
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DNodeArray **rootnodes. The returned array of root Nodes that comprise this context.

The `getRootNodes` service returns a listing of the current root nodes of the execution context. If the context was generated from a file, the root nodes are in the order they were declared in the file. Any added nodes are then appended to the list in the time order they were received at the browser. If the context was generated programmatically, the nodes are in the order they were received by the browser.

6.4.17 dispose

HRESULT print (void);
Return Values: S_OK, E_FAIL
Parameters: none

The `dispose` service specifies that the client has no further interest in the resource represented by this execution context. The browser may now take whatever action is necessary to reclaim any resources consumed by this execution context, now or at any time in the future. If this execution context has already been disposed, further requests have no effect.

6.5 Scene services

6.5.1 Introduction

A scene is an extension of the execution context services with additional services provided. The Scene services implementation shall include all of the services from section 6.4, above, and include the following additional services.

6.5.2 getMetadata

HRESULT getMetaData (
/* [in] */ BSTR key,
/* [retval][out] */ BSTR *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR key. The kunicode key string used to find the MetaData item.
/* [retval][out] */ BSTR *value. The returned value of the specified metatdata.

The `getMetadata` service returns an item of metadata from the scene that was specified using the META statement defined in 7.2.5.5 META statement of Part 1 of ISO/IEC 19775 (see 2.[I19775-1]). Metadata specified in the META statement is represented as an SAIString key/value pair. Each key corresponds to exactly zero or one value. Optionally, the browser may provide a subservice to discover the valid keys for this scene as part of this service.
Metadata defined by metadata nodes as defined in Part 1 of ISO/IEC 19775 can be manipulated using 6.6 Node services.

6.5.3 setMetadata

HRESULT setMetaData (
/* [in] */ BSTR key,
/* [in] */ BSTR value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ BSTR key. The unicode key string used to define the metaData item.
/* [in] */ BSTR value. The value to be set, in unicode format.

The `setMetadata` service inserts an item of metadata in the scene in the form of a META statement as defined in 7.2.5.5 META statement of Part 1 of ISO/IEC 19775 (see 2.[I19775-1]). Metadata is represented as a SAIString key/value pair. Each key corresponds to exactly zero or one value. Setting an item with a key that already exists replaces the existing value. If the value is `NULL` for the given key, the META statement associated with that key is removed from the scene.
Metadata defined by metadata nodes as defined in Part 1 of ISO/IEC 19775 can be manipulated using 6.6 Node services.

6.6 Node services

6.6.1 Introduction

The following services can be requested of an individual node. Each service requires an identifier for that node. After a request of an individual node to dispose of their resources, any further request made to a node service shall generate a disposed error.
Although not specified, all services are capable of throwing an SAI_CONNECTION_ERROR whenever a request is made if the session between the application and the browser has failed.

6.6.2 getTypeName

```
HRESULT getTypeName (
        /* [retval][out] */ BSTR *typenm);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR *typenm.  The returned type name of the node, in Unicode.
```

The `getTypeName` service returns the name of the type of the referenced node. The type name is the name as specified in ISO/IEC 19775-1 where the node type is defined (see *Node index* in part 1 of ISO/IEC 19775 (2.[I19775-1]) for easy access to a node definition). If the node represents a PROTO node instance, the type name returned is the name of the PROTO declaration.

6.6.4 getField

```
HRESULT getField (
        /* [in] */ BSTR name,
        /* [retval][out] */ X3DField **result);
Return Values: S_OK, E_FAIL
Parameters:
        /* [in] */ BSTR name. The name of the field to be returned.
        /* [retval][out] */ X3DField **result.  The returned pointer to the desired field.
```

The `getField` service returns a field identifier so that operations can be performed on the node properties. If the field requested is an inputOutput field, either the field name or the set_ and _changed modifiers may be used to access the appropriate form of the node as required. Access to fields is implementation dependent.

6.6.5 getFieldDefinitions

HRESULT getFieldDefinitions (
/* [retval][out] */ X3DFieldDefinitionArray **fieldDefinitions);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DFieldDefinitionArray **fieldDefinitions. The returned array of Field Definition objects.

The `getFieldDefinitions` service returns a list of all the field definitions of the referenced node. The definitions provide a limited form of the SAIField that has all the same services except the ability to read or write the value of the field for a specific node instance. This request returns the SAIField values as generic responses for every instance of this node rather than for a specific instance.

6.6.6 dispose

HRESULT dispose (void);
Return Values: S_OK, E_FAIL
Parameters: none

The `dispose` node service indicates that the client has no further interest in the resource represented by this node. The browser may take whatever action is necessary to reclaim any resources consumed by this node, now or at any time in the future. If this node has already been disposed, further requests have no effect.
Disposing of a node does not remove the node from the scene graph (if it was inserted in the first place) but rather removes any local information per client to it. The underlying X3D node representation is only disposed if no other applications or scene graph structures contain references to this node and the responsibility and timing for this action is browser implementation specific.

6.6.7 getNumFields

HRESULT getNumFields (
/* [retval][out] */ int *result);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ int *result. The number of fields attached to the Node definition.

The getNumFields service returns the number of Fields contained by the node.

6.6.8 getFieldInfo

```
HRESULT getNumFields (
        /* [in] */ int fieldIndex,
        /* [out] */ X3DFieldAccess *accessType,
        /* [out] */ X3DFieldType *fieldType,
        /* [out] */ BSTR *fieldName);
```

Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int fieldIndex. Index of the field, used to identify the field on the Node.
/* [out] */ X3DFieldAccess *accessType. The AccessType of the field.
/* [out] */ X3DFieldType *fieldType. The Type of the Field.
/* [out] */ BSTR *fieldName. The name of the field, in Unicode.

The getFieldInfo method allows you to get the information about the field, given its index. Using this method, and the getNumFields method, you can traverse the fields of any Node, and get at all the field values.

6.7 Field services

6.7.1 Introduction

The following are services that can be requested of individual fields of a node. If the node from which a field was retrieved has been disposed, field services are still permitted to operate providing that the field reference has been obtained before disposing of the node. If a call is made to a field service after requesting disposal of the field, a disposed error shall be generated.
Although not specified, all services are capable of throwing an SAI_CONNECTION_ERROR whenever a request is made if the session between the application and the browser has failed.

6.7.2 getAccessType

```
HRESULT getAccessType (
```

/* [retval][out] */ X3DFieldAccess *accesstype);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DFieldAccess *accesstype. The returned accessType of the field.

The `getAccessType` service returns the access type for the specified field of the referenced node.

6.7.3 getType

HRESULT getType (
/* [retval][out] */ X3DFieldType *fieldtype);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DFieldType *fieldtype. The returned Field Type.

The `getType` field service returns the type for the specified field of the referenced node.

6.7.4 getName

HRESULT getName (
/* [retval][out] */ BSTR *name);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR *name. The returned Field Name.

If supported by the implementation, the `getName` field service returns the name of the field as it was requested from the node. If the service requested the *set_children* field of a grouping node, this shall return "set_children", but if a different request was for *children* on the same node, "children" shall be returned.

6.7.5 getValue

See Field Specific versions of these interfaces below.

The `getValue` field service returns the value represented by the specified field as it exists in the world. This represents the current value of the field at the time of the request. If the request is made of a field that has a setValue request buffered through `BeginUpdate`, the value returned shall be the old value prior to the setValue request. The value of the

field may be a node if the field represents an MFNode or SFNode.
All field types shall support the option to return a single value from multi-valued arrays.

6.7.6 setValue

See Field Specific versions of these interfaces below.

The `setValue` field service sets the value of the specified field. Set requests shall obey the requirements as specified for buffered events services.
The value of the field may be an SAINode value if the field represents an MFNode or SFNode. It is permitted to send a null to a node or field in order to clear the value from that field. For example sending a null to the appearance inputOutput field of a Shape node as specified in *12 Shape component* in part 1 of ISO/IEC 19775 (see 2.[I19775-1]), shall result in the *appearance* field being cleared and set to the default value of NULL.
If the SAINode value is registered as an IMPORTed node in this file, it shall generate the SAI_IMPORTED_NODE error.
All field setting services implementations shall include the ability to set individual values. Fields that describe multi-value arrays shall also include the ability to append and remove items from the existing field.

6.7.7 registerFieldInterest

HRESULT addFieldEventListener (
/* [in] */ IDispatch *listener);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ IDispatch *listener. The listener that gets called when the field value is set.

HRESULT removeFieldEventListener (
/* [in] */ IDispatch *listener);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ IDispatch *listener. Removed this listener for the ListenerList.

The `registerFieldInterest` service nominates the requester as the receiver of all SAIFieldEvents. The act of making this service request itself does not imply any events shall be generated.

The parameter of type SAIRequester can be inferred from the source of the input and may not need to be part of the parameters.
The parameter of type SAIAction specifies whether this is a request to add interest in events or to remove interest in the events.
Which capabilities are permitted to be listened to are implementation dependent. For example, some implementations may permit listening to inputOnly values and outputOnly values while others will only permit listening to outputOnly values.
For SFNode and MFNode field types, the following additional behaviour is specified:

> Nodes are bound by the capabilities of the containing scene. No node shall be of greater capabilities than the scene's declared profile and additional components.
> SAI_INSUFFICIENT_CAPABILITIES shall be generated if the action is to add a node to the scene and that node requires greater capabilities than the scene permits.

If the action is to add a node, and the node or any of its children is currently part of another scene, an SAI_NODE_IN_USE error shall be generated.
If the action is to remove a node and the node is not a known value of this field, the request shall be silently ignored.

6.7.8 dispose

HRESULT dispose (void);
Return Values: S_OK, E_FAIL
Parameters: none

The `dispose` field service indicates that the client has no further interest in the resource represented by this field. The browser may take whatever action is necessary to reclaim any resources consumed by this field, now or at any time in the future. If this field has already been disposed, further requests have no effect.

6.8.0 Specific Field Types:

Specific Field Types:
The following interfaces inherit from the X3Dfield class.
There is one field class for each type of Field.

The Single Value fields are defined, followed my the multiple value fields.

All fields inherit the X3Dfield interface defined above.
All multiple value fields also inherit the X3DMField interface declared at the top of the multiple value fields.

```
X3DSFBool
   HRESULT  getValue(
      /* [retval][out] */ BOOL *value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [retval][out] */ BOOL *value);

   HRESULT  setValue(
      /* [in] */ BOOL value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [in] */ BOOL value);



X3DSFColor
   HRESULT  getValue(
      /* [size_is][out][in] */ float *value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [size_is][out][in] */ float *value);

   HRESULT  setValue(
      /* [size_is][in] */ float *value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [size_is][in] */ float *value);

   HRESULT  get_r(
      /* [retval][out] */ float *pVal);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [retval][out] */ float *pVal);
```

```
    HRESULT  put_r(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT  get_g(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_g(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT  get_b(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_b(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);




X3DSFFloat
    HRESULT  getValue(
        /* [retval][out] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *value);

    HRESULT  setValue(
        /* [in] */ float value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float value);


X3DSFInt32
    HRESULT  getValue(
        /* [retval][out] */ long *value);
            Return Values: S_OK, E_FAIL
```

```
            Parameters:
/* [retval][out] */ long *value);

    HRESULT  setValue(
        /* [in] */ long value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ long value);

X3DSFNode
    HRESULT  getValue(
        /* [retval][out] */ X3DNode **value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ X3DNode **value);

    HRESULT  setValue(
        /* [in] */ X3DNode *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ X3DNode *value);

X3DSFRotation
    HRESULT  getValue(
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [size_is][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][in] */ float *value);

    HRESULT  get_x(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_x(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT  get_y(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
```

/* [retval][out] */ float *pVal);

    HRESULT put_y(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT get_z(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT put_z(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT get_angle(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT put_angle(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

X3DSFString
    HRESULT getValue(
        /* [out][in] */ BSTR *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [out][in] */ BSTR *value);

    HRESULT setValue(
        /* [in] */ BSTR value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ BSTR value);

X3DSFVec2f
    HRESULT getValue(

```
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [size_is][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][in] */ float *value);

    HRESULT  get_x(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_x(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT  get_y(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_y(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);



X3DSFVec3f
    HRESULT  getValue(
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [size_is][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][in] */ float *value);

    HRESULT  get_x(
```

```
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_x(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT  get_y(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_y(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

    HRESULT  get_z(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ float *pVal);

    HRESULT  put_z(
        /* [in] */ float newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ float newVal);

X3DSFVec4f
    HRESULT  getValue(
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [size_is][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [size_is][in] */ float *value);

    HRESULT  get_x(
        /* [retval][out] */ float *pVal);
            Return Values: S_OK, E_FAIL
```

Parameters:
/* [retval][out] */ float *pVal);

HRESULT put_x(
/* [in] */ float newVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ float newVal);

HRESULT get_y(
/* [retval][out] */ float *pVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ float *pVal);

HRESULT put_y(
/* [in] */ float newVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ float newVal);

HRESULT get_z(
/* [retval][out] */ float *pVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ float *pVal);

HRESULT put_z(
/* [in] */ float newVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ float newVal);

HRESULT get_w(
/* [retval][out] */ float *pVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ float *pVal);

HRESULT put_w(
/* [in] */ float newVal);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ float newVal);

Multiple Value Fields:

X3DMField
HRESULT get_Count(

```
        /* [retval][out] */ long *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ long *pVal);

    HRESULT  get_length(
        /* [retval][out] */ long *pVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ long *pVal);

    HRESULT  put_length(
        /* [in] */ long newVal);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ long newVal);

    HRESULT  Item(
        /* [in] */ VARIANT index,
        /* [retval][out] */ VARIANT *pItem);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ VARIANT index,
/* [retval][out] */ VARIANT *pItem);


X3DMFColor
    HRESULT  getValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  set1Value(
        /* [in] */ int index,
        /* [in] */ X3DSFColor *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [in] */ X3DSFColor *value);
```

```
   HRESULT  get1Value(
      /* [in] */ int index,
      /* [retval][out] */ X3DSFColor **value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [in] */ int index,
/* [retval][out] */ X3DSFColor **value);



X3DMFNode
   HRESULT  getValue(
      /* [retval][out] */ X3DNodeArray **value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [retval][out] */ X3DNodeArray **value);

   HRESULT  setValue(
      /* [in] */ X3DNodeArray *value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [in] */ X3DNodeArray *value);

   HRESULT  get1Value(
      /* [in] */ int index,
      /* [out] */ X3DNode **value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [in] */ int index,
/* [out] */ X3DNode **value);

   HRESULT  get1ValueV(
      /* [in] */ int index,
      /* [retval][out] */ VARIANT *value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [in] */ int index,
/* [retval][out] */ VARIANT *value);

   HRESULT  set1Value(
      /* [in] */ int index,
      /* [in] */ X3DNode *value);
         Return Values: S_OK, E_FAIL
         Parameters:
/* [in] */ int index,
/* [in] */ X3DNode *value);



X3DSFTime
   HRESULT  getValue(
```

```
        /* [retval][out] */ double *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [retval][out] */ double *value);

    HRESULT  setValue(
        /* [in] */ double *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ double *value);



X3DMFFloat
    HRESULT  getValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  set1Value(
        /* [in] */ int index,
        /* [in] */ float value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [in] */ float value);

    HRESULT  get1Value(
        /* [in] */ int index,
        /* [retval][out] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [retval][out] */ float *value);
```

X3DMFInt32
    HRESULT getValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ long *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ long *value);

    HRESULT setValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ long *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ long *value);

    HRESULT set1Value(
        /* [in] */ int index,
        /* [in] */ long value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [in] */ long value);

    HRESULT get1Value(
        /* [in] */ int index,
        /* [retval][out] */ long *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [retval][out] */ long *value);

X3DMFRotation
    HRESULT getValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ long *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ long *value);

    HRESULT setValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL

Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

HRESULT set1Value(
/* [in] */ int index,
/* [in] */ X3DSFRotation *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [in] */ X3DSFRotation *value);

HRESULT get1Value(
/* [in] */ int index,
/* [retval][out] */ X3DSFRotation **value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [retval][out] */ X3DSFRotation **value);

X3DMFString
HRESULT getValue(
/* [in] */ int cnt,
/* [size_is][out][in] */ BSTR *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ BSTR *value);

HRESULT setValue(
/* [in] */ int cnt,
/* [size_is][out][in] */ BSTR *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ BSTR *value);

HRESULT set1Value(
/* [in] */ int index,
/* [in] */ BSTR value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [in] */ BSTR value);

HRESULT get1Value(
/* [in] */ int index,
/* [retval][out] */ BSTR *value);

Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [retval][out] */ BSTR *value);

X3DMFVec2f
HRESULT getValue(
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

HRESULT setValue(
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

HRESULT set1Value(
/* [in] */ int index,
/* [in] */ X3DSFVec2f *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [in] */ X3DSFVec2f *value);

HRESULT get1Value(
/* [in] */ int index,
/* [retval][out] */ X3DSFVec2f **value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [retval][out] */ X3DSFVec2f **value);

X3DMFVec3f
HRESULT getValue(
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

```
    HRESULT  setValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  set1Value(
        /* [in] */ int index,
        /* [in] */ X3DSFVec3f *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [in] */ X3DSFVec3f *value);

    HRESULT  get1Value(
        /* [in] */ int index,
        /* [retval][out] */ X3DSFVec3f **value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [retval][out] */ X3DSFVec3f **value);



X3DMFVec4f
    HRESULT  getValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  setValue(
        /* [in] */ int cnt,
        /* [size_is][out][in] */ float *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int cnt,
/* [size_is][out][in] */ float *value);

    HRESULT  set1Value(
        /* [in] */ int index,
        /* [in] */ X3DSFVec4f *value);
            Return Values: S_OK, E_FAIL
            Parameters:
/* [in] */ int index,
/* [in] */ X3DSFVec4f *value);
```

HRESULT get1Value(
/* [in] */ int index,
/* [retval][out] */ X3DSFVec4f **value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ int index,
/* [retval][out] */ X3DSFVec4f **value);

X3DSFDouble
HRESULT getValue(
/* [out][in] */ double *value);
Return Values: S_OK, E_FAIL
Parameters:
/* [out][in] */ double *value);

HRESULT setValue(
/* [in] */ double value);
Return Values: S_OK, E_FAIL
Parameters:
/* [in] */ double value);

6.8 Route services

6.8.1 getSourceNode

HRESULT getSourceNode (
/* [retval][out] */ X3DNode **pSourceNode);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DNode **pSourceNode. Returns a pointer to the Source node of this ROUTE.

The `getSourceNode` service returns the source node of the specified route.

6.8.2 getSourceField

HRESULT getSourceField (
/* [retval][out] */ BSTR *sourcefield);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR *sourcefield. The return value of the source field name, in unicode.

The `getSourceField` service returns the name of the source field of the specified route.

6.8.3 getDestinationNode

HRESULT getDestinationNode (
/* [retval][out] */ X3DNode **pDestinationNode);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ X3DNode **pDestinationNode. Returns a pointer to the Destination node of this ROUTE.

The `getDestinationNode` service returns the destination node of the specified route.

6.8.4 getDestinationField

HRESULT get DestinationField (
/* [retval][out] */ BSTR * destinationfield);
Return Values: S_OK, E_FAIL
Parameters:
/* [retval][out] */ BSTR * destinationfield. The return value of the source field name, in unicode.

The `getDestinationField` service returns the name of the destination field of the specified route.

6.8.5 dispose

HRESULT dispose (void);
Return Values: S_OK, E_FAIL
Parameters: none

The `dispose` route service indicates that the client has no further interest in the resource represented by this route. The browser may take whatever action is necessary to reclaim any resources consumed by this route, now or at any time in the future. If this route has already been disposed, further requests have no effect.
Disposing of a route does not remove the route from the scene graph (if it was inserted in the first place) but rather removes any local information per client to it. The underlying X3D node representation is only disposed of if no other applications or scene graph structures contain references to

this route and the responsibility and timing for this action is browser implementation specific.

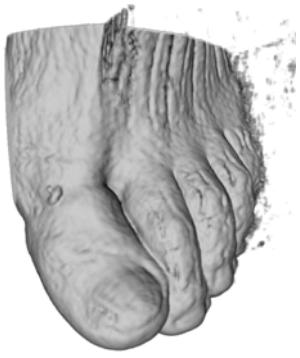
Appendix I

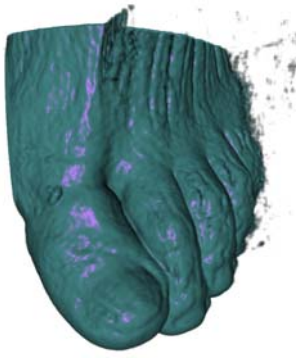
Illustrations of Volume Rendering Styles in MedX3D Browser

OpacityMapVolumeStyle

PhongVolumeStyle

- Phong-Blinn shading
  - Ambient, diffuse, specular

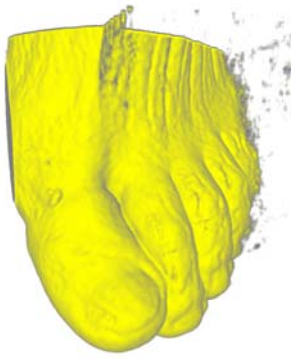




Non-photorealistic Styles

- CartoonVolumeStyle
  - Two colours, discrete sections
- ToneMappedVolumeStyle
  - Cool and warm color

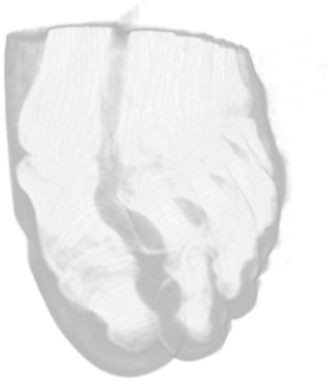
Cartoon



ToneMapped

Enhancement styles

- EdgeEnhancementVolumeStyle
- BoundaryEnhancementVolumeStyle
- SilhouetteEnhancementVolumeStyle

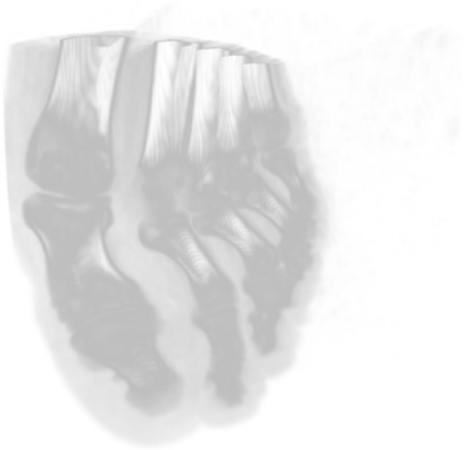


Boundary

Silhouette

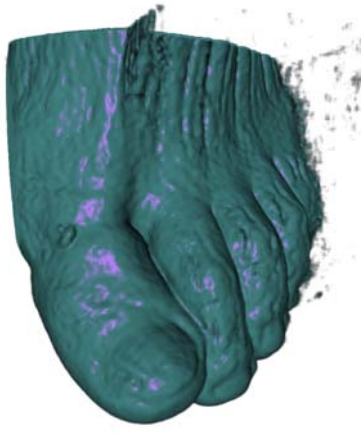
MIPVolumeStyle

- Maximum Intensity Projection
  - Maximum intensity along each ray

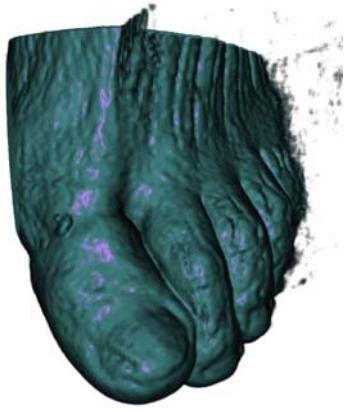


ComposedVolumeStyle

- Compose several styles into one



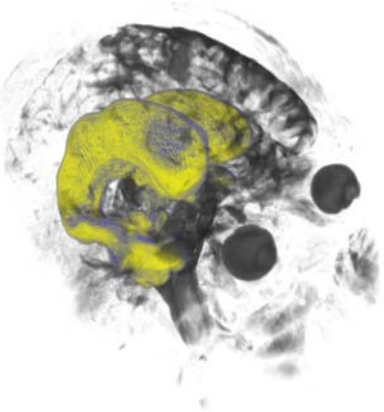
Opacity-Phong



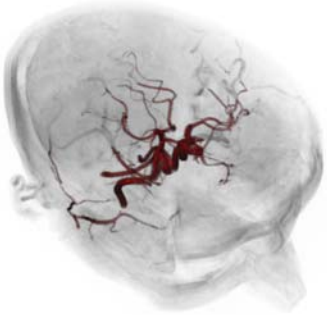
Opacity-Phong-Edge enhancement

SegmentedVolumeData

- Different parts of a dataset are rendered with different rendering styles.



Cerebral ventricular system: Tone Map
Head: Opacity Map



Blood vessels: Phong
Head: Opacity Map-Boundary Enhancement

MedX3D: Standards Enabled Desktop Medical 3D

N.W. John, M. Aratow, J. Couch, D. Evestedt, A.D. Hudson, N. Polys, R.F. Puk, A. Ray, K. Victor, Q. Wang
The Web3D Consortium

Abstract. This paper reports on the work of the Web3D Consortium's Medical Working Group to specify and implement MedX3D – an extension to the X3D standard that will support advanced medical visualization functionality and medical data exchange. This initiative covers volume rendering, ontology support, and data import/export, for standalone applications and web-based plug-ins. It is our hypothesis that such a 3D medical standard will provide better access to data, and enable improvements in medical care.

Keywords. X3D, medical visualization, volume rendering, Web applications

1. Background/Problem

The capability to deliver three dimensional (3D) visualizations of patient medical scan data is well established. Today, medical scanner manufacturers and independent companies routinely provide workstations that support 3D and such workstations are becoming faster and the software more sophisticated. However, accessibility to this 3D technology needs to increase before the medical profession will see significant efficiency gains and benefits to patients [1]. The use of Web technologies provides an attractive solution for providing good accessibility and interoperability, and there have been several web-based uses of medical visualization applications reported in the last decade [2]. The ISO standard for using 3D graphics over the internet is X3D [3] and the Web3D Consortium has formed a Medical Working Group with a remit to develop an extension to X3D, called MedX3D, which will support the required functionality and interoperability needed for any medical visualization application. Such a 3D medical standard will enable improvements in medical care, including: enhanced informed consent; surgical planning and performance; medical education; and accessibility.

This paper reports on the results of recently completed projects funded by the US Army's Telemedicine and Advanced Technology Research Center (TATRC) to specify and implement the first version of MedX3D.

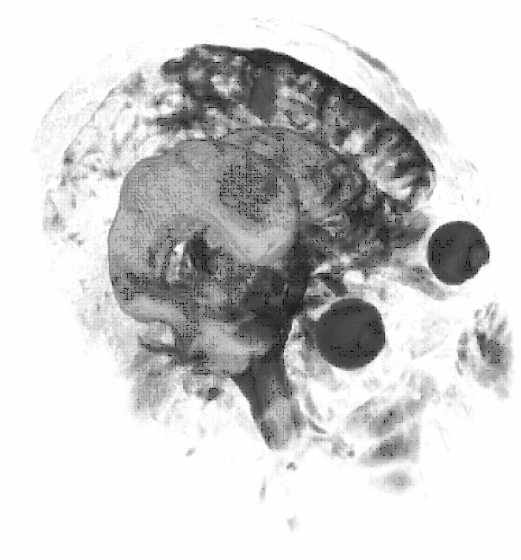


Figure 1: VRE example: Cerebral ventricular system. Tone Mapped Head. Using Opacity

2. Tools and Methods

This work has been an international collaborative initiative drawing on expertise from industry, academia and hospitals. The main contributors have been Intelligraphics, Inc. (USA), Media Machines, Inc. (USA), NIST (USA), SenseGraphics AB (Sweden), VirtuWorlds (USA), Yumetech, Inc. (USA), and Bangor University (UK). Four key tasks have been completed:

1. Specification of both the X3D Volume Rendering Extension (VRE) and the MedX3D profile.
2. Provision of support for anatomical ontologies.
3. An import/export library for MedX3D.
4. A MedX3D web browser plug-in that can read any DICOM data set and provide 3D visualization through the VRE.

These tasks are described in more detail below.

2.1. Specification of the X3D Volume Rendering Extension and MedX3D

The possibility of volume rendering in a web browser was first demonstrated by Hendin [4] using the VRML 97 standard. Performance has improved dramatically since then largely due to the developments in PC graphics card technology. Volume rendering support is now a core requirement for 3D medical visualization and so an early task in enabling MedX3D was to provide a specification for the X3D Volume Rendering Extension (VRE) in a format ready for ISO ratification. There are many

different techniques for rendering volumetric data including: plane slicing; the use of real time shaders; and ray tracing. The VRE does not define the technique used to render the data, however, only the type of visual output needed. Rendering nodes are used to define the outputs, and a full implementation of the VRE specification will require support for: boundary, edge and silhouette enhancement; opacity maps; isosurfaces; cartoon-style non-photorealistic rendering; stipple patterns; and compositing multiple styles together into a single rendering pass. Figure 1 shows our implementation of the VRE in action using the tone mapped volume style - an implementation of the Gooch shading model of two-toned warm/cool colouring [5]. Volumetric data representation in the VRE makes use of the X3D 3D texturing component [6].

MedX3D builds on the current X3D specification and the VRE to define a medical interchange profile. A series of user cases were compiled, and combined with a literature search to determine what the profile should contain. It has been optimized to support medical visualization and markup within an X3D compliant browser or application. The use-cases were developed with DICOM and are designed to bridge the success DICOM has had with equipment manufacturers with the success Web3D has had in getting 3D graphics onto the web.

2.2. Ontology Support

Two ontologies have been used during this work: the FMA (Foundational Model of Anatomy) [7], and SNOMED CT® (Systematized Nomenclature of Medicine-Clinical Terms). They provide comprehensive computer-based knowledge sources in the biomedical sciences and clinical healthcare. Table 1 summarizes the components of both sources and a detailed comparison can be found in [8].

FMA	SNOMED CT®
75,000 classes, 120,000 terms	357,000 concepts with unique meanings and formal logic-based definitions organized into hierarchies
2.1 million relationship instances from 168 relationship types link the FMA's classes into a coherent symbolic model	Core general terminology for the electronic health record (EHR)
Part of the Anatomy Information System developed at the University of Washington	Joint development between the UK NHS and the College of American Pathologists

Table 1: Overview of FMA and SNOMED ontologies

To support an ontology in X3D, we need a method for semantic medical data to be incorporated into the scenegraph via the use of the Metadata nodeset. This involved finding an appropriate representations of ontology information in X3D Metadata, creating transformation mechanisms for the information, evaluating their similarities, and showing examples of the metadata in use (Figure 2). The integration of these ontologies with X3D scenes (such as anatomical models) provides the foundation for

interactive 3D objects to be represented with all their attributes and relationships during run time. Through the lossless transformation mechanisms we have demonstrated that concepts and relationships, and systems and parts can all be cross-referenced within an X3D application or through an external application. Figure 2 is a snapshot of a simple application running in an X3D browser that demonstrates this functionality. The taxonomy data displayed is updated (using the FMA and SNOMED) each time a different organ is selected.

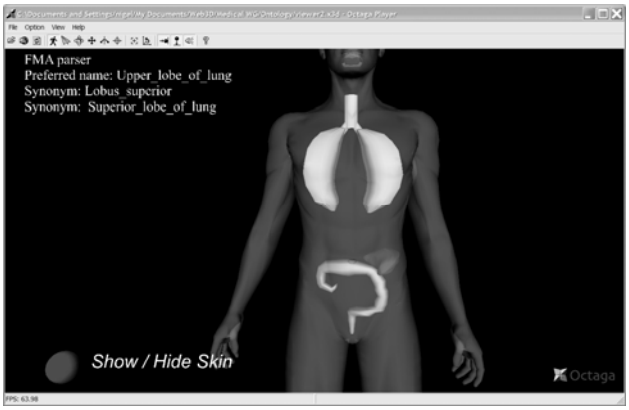


Figure 2. As the user selects each organ (mouse click) in this simple 3D Anatomical model displayed within an X3D browser, relevant taxonomy from the FMA and SNOMED is displayed

2.3. Import/Export Library

The goal of import/export library is to enable other software to interact with MedX3D content. As well as providing the ability to import and export content, the library supports an API that gives developers access to the scene graph. Thus content can be generated programmatically and subsequently exported. The API also allows imported content (that is compliant with the MedX3D profile) to be interrogated.

The development utilized the Flux X3D-based open-source engine [9]. The import/export API is closely aligned with the Scene Access Interface (SAI) [10], which allows an application programmer to inspect or build X3D content. Microsoft's COM was used for the library infrastructure. The API supports two different languages, C++ and Visual Basic. The API is fully documented.

The installation also includes two test harness applications. One in Visual Basic, and one in C++. The C++ test harness application shows how the API can be used to import, and interrogate an X3D file, by displaying the node hierarchy of the contents in a GUI tree window. The test harness then programmatically adds content to the scene, and exports the result, thus illustrating the ability to programmatically generate X3D content.

2.4. MedX3D Plug-in

A web browser plug-in has been implemented to support the new MedX3D profile including the VRE. It supports reading DICOM data sets and the wide variety of volume styles available in the VRE can be used to visualize the data (only the StippleVolumeStyle is not yet implemented). The VRE specification was written to be implementation independent so as not to be locked into a specific volume rendering technique. The plug-in supports two different volume rendering techniques: GPU-based ray casting; and slices using 3D-textures. The development utilized the X3D-based open-source development platform H3D API [11] and was implemented using OpenGL and GLSL shaders. The plug-in supports multiple browsers and makes it possible to visualize and explore volume data using Internet Explorer, Mozilla Firefox (Figure 3) or Opera.

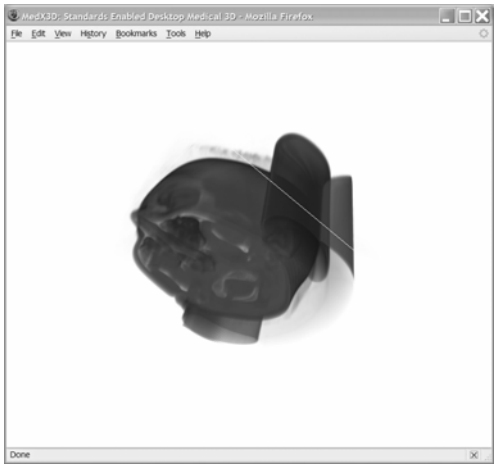


Figure 3. The MedX3D plug-in running inside Mozilla Firefox. A CT data set is being volume rendered in real time by automatically taking advantage of the GPU.

3. Results

A completely specified volume rendering extension and profile for X3D has now been prepared for final ISO ratification - this is the core of MedX3D. Figure 3 shows the MedX3D plug-in being used within Mozilla Firefox to explore a 3D opacity map volume rendering of a CT data set. Ontology data can be accessed when available and seamlessly integrated into the 3D scene. A sophisticated medical visualization application can quickly be built by using the MedX3D profile. The main advantage of MedX3D is that it enables convenient access to DICOM image data from within a Web browser whilst providing the same quality and key visualization and markup features available from a dedicated 3D medical visualization workstation. We are currently carrying out evaluation studies to compare the ease of use and rendering quality of the standards-based MedX3D approach with currently available commercial and public domain software.

4. Conclusions/Discussion

By utilizing an ISO ratified, open and royalty free standard, the necessary prerequisites for a standard, archive file format for 3D medical images have been achieved. In addition, we have shown that it is possible to integrate both FMA and SNOMED information individually into an X3D scenegraph with a lossless transformation. It is theoretically possible for both FMA and SNOMED transformations to be combined together in one single mapping tool handling both data sources.

The results of this project may constitute Recommended Practice for the Web3D Consortium's Medical Working Group and inform the development of semantically-integrated interactive 3D applications (i.e. anatomy browsers and imaging (DICOM) tool vendors). A common scenario would be to use these conventions to semantically 'tag' segmented anatomical structures for storage or delivery as an X3D environment.

Discussions are also ongoing with the DICOM WG-17 (3D) who are responsible for extending the DICOM Standard with respect to 3D and other multi-dimensional data sets. We are investigating whether MedX3D can become the enabling technology for this purpose.

References

[1] Megibow, A.J. (2002). Three-D offers workflow gains, new diagnostic options. Diagnostic Imaging. November 2002, 83-93

[2] John, N.W. (2007). The Impact of Web3D Technologies on Medical Education and Training. Computers & Education. Vol 49. Issue 1, August 2007, 19-31

[3] Brutzman, D. and Daly, L. (2007). X3D Extensible 3D Graphics for Web Authors. The Morgan Kaufmann Series in Computer Graphics. ISBN 978-0-12-088500-8

[4] Hendin, O., John, N.W., Shochet, O. (1998). Medical Volume Rendering on the WWW using JAVA and VRML. Stud Health Technol Inform 50, 34-40

[5] Gooch, A., Gooch, B., Shirley, P., and Cohen, E. (1998) A non-photorealistic lighting model for automatic technical illustration. In Proceedings of the 25th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '98. ACM Press, New York, NY, 447-452.

[6] X3D 3D Texturing Component http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification_Revision1_to_Part1/Part01/components/texture3D.html Last visited October 2007

[7] Rosse C., Mejino J.V.L. (2003). A reference ontology for biomedical informatics: the Foundational Model of Anatomy. J Biomed Inform. 36:478-500.

[8] Bodenreider, O. and Zhang, S. (2006). Comparing the Representation of Anatomy in the FMA and SNOMED CT. AMIA Annu Symp Proc. 2006; 46–50

[9] http://sourceforge.net/projects/flux Last visited October 2007

[10] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/ Last visited October 2007

[11] H3D Web Site, http://www.h3d.org/ Last visited October 2007

Acknowledgements

We wish to thank The Web3D Consortium for providing the infrastructure and expertise to make this work possible, and TATRC for providing the funding.