Enriching PDF 3D documents with semantic metadata from multiple sources

Gerald Buchgraber

Enriching PDF 3D documents with semantic metadata from multiple sources

Master's Thesis

at

Graz University of Technology

submitted by

Gerald Buchgraber

Institute of Computer Graphics and Knowledge Visualization,
Graz University of Technology
A-8010 Graz, Austria

10th May 2010

Advisor: Dipl.-Inform. René Berndt
Supervisor: Dipl.-Inform. Dr.-Ing. Sven Havemann

Verknüpfung von 3D Inhalten mit unterschiedlichen semantischen Metadaten innerhalb von PDF Dokumenten

Masterarbeit

an der

Technischen Universität Graz

vorgelegt von

Gerald Buchgraber

Institut für Computer Graphik und Wissensvisualisierung,
Technische Universität Graz
A-8010 Graz

10. Mai 2010

Diese Arbeit ist in englischer Sprache verfasst.

Betreuer:	Dipl.-Inform. René Berndt
Begutachter:	Dipl.-Inform. Dr.-Ing. Sven Havemann

Abstract

Three-dimensional content is useful in a great variety of real-world applications, not only for technical drawings or computer games, but also for scientific documentation to provide a better understanding of inherently multi-dimensional data in comparison to two-dimensional graphics. Apart from just displaying a static object, three-dimensional visualizations can significantly be enriched by semantic metadata, in form of further textual information for specific parts, predefined views, dynamic textures or additional 3D objects that can be introduced and manipulated programmatically, possibly in response to a custom user interaction.

Based on the ubiquitous availability of Adobe Reader on multiple platforms and the support for directly embedding 3D objects in PDF documents, one of the main obstacles in delivering 3D content to a broader audience seems to be solved. However, the generation of such 3D-contained PDF documents is not trivial and in most cases still requires commercial tools like Adobe Acrobat.

The approach that is presented in this thesis is exclusively based on open standards and open-source software. It relies on XML technology and proposes a standard compliant extension to XSL-FO, for enabling an FO document to include 3D content, especially for generating PDF documents. This work presents a lean, flexible and also extensible solution, which can be integrated with existing document workflow technology, in desktop or (web)server environments. Two-dimensional documentation workflows can therefore easily be extended to contain rich and interactive 3D visualizations.

Kurzfassung

Der Einsatz von dreidimensionalen Inhalten ist, abgesehen von technischen Modellen und Computerspielen, auch für eine ganze Reihe von weiteren Anwendungsbereichen, wie wissenschaftlichen Publikationen und Visualisierungen sehr gut geeignet. Speziell für die Darstellung mehrdimensionaler Daten aus nahezu allen Wissensbereichen ergeben sich, gegenüber einer zweidimensionalen Grafik, deutliche Vorteile die bei Betrachtern zu einem besseren Verständnis führen. Abgesehen von der reinen Darstellung eines dreidimensionalen Modells, können zusätzliche Meta-Informationen in Form von vordefinierten Ansichten, dynamischen Texturen, weiteren 3D Objekten, beschreibenden Texten im Raum und einer beliebig gestalteten Benutzerinteraktion zu einem viel reichhaltigerem 3D Erlebnis führen.

Seit dem Jahr 2005 ist es möglich solche 3D Inhalte vollständig in PDF Dokumente einzubetten und durch die enorme Benutzerakzeptanz gegenüber dem PDF Betrachtungsprogramm Adobe Reader, können diese auch auf den meisten Betriebssystemen ohne Installation von zusätzlichen Software-Paketen betrachtet werden. Die Erstellung solcher Dokumente gestaltet oftmals als schwierig und beruht hauptsächlich auf kommerziellen Produkten wie Adobe Acrobat.

In dieser Masterarbeit wird ein neuer Ansatz für das Erstellen von PDF Dokumenten, die dreidimensionale Inhalte und damit verbundene Meta-Informationen enthalten, vorgestellt. Dieser Ansatz basiert auf offenen XML Standards und erweitert XSL-FO im Rahmen der W3C Spezifikation um die Möglichkeit auch dreidimensionale Objekte, in erster Linie für die Integration in PDF Dokumenten, repräsentieren zu können. Da XML, wie auch XSL-FO, gängige und häufig verwendete Standards sind, lässt sich der vorgestellte Ansatz auch sehr gut in bereits bestehende Dokumentationsabläufe integrieren und kann sowohl in Desktop- als auch Serverumgebungen verwendet werden.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank all people who have helped and inspired me during the time of my study and finally supported me while writing this thesis.

First of all, my advisor René Berndt, supervisor Sven Havemann and Martin Strobl, from the Institute of Computer Graphics and Knowledge Visualization at Graz University of Technology, deserve a big thank-you for their technical guidance and their pleasant support.

I want to thank my fellow students and very good friends Gernot Margreitner and René Ranftl for having countless coffee breaks and numerous valuable discussions.

All this would not have been possible without the unconditional support of my family, especially by my parents Anna Maria and Johann Buchgraber who enabled me to study.

Most of all, I want to thank my girlfriend and soon-to-be wife Katharina for her patience throughout the whole study, constant encouragement when it was most required and her long-lasting love.

Gerald Buchgraber
Graz, Austria, May 2010

Credits

I would like to thank René Berndt, Graz University of Technology, for permission to use his 3D-chessman models (Section 6.3). Furthermore, this thesis was written using the LaTeX skeleton thesis from Keith Andrews [10].

Chapter 1

Introduction

With the tremendously fast growing 3D graphics market and the associated price decline in 3D graphics hardware, today even low-cost computers are capable of displaying 3D content in realtime. Nowadays, not only conventional computers, but also mobile devices are able to present and interact with virtual 3D objects. One could almost say that 3D support is omnipresent. However, the overwhelming majority of users is neihter aware of 3D graphics support in their devices, nor are they confronted with interactive digital 3D content at all.

In January 2005, Adobe Systems Incorporated published version 1.6 of their popular and widely adopted *Portable Document Forma*t (PDF). In the specification of PDF 1.6 and the corresponding version of Acrobat 7, support for embedded 3D content was added. In the mean time the subsequent version PDF 1.7 has already reached the status of an ISO standard [4]. Since PDF nowadays is a global and especially trusted standard for electronic documents, almost all PC users are familiar with such documents and a corresponding PDF viewer application. Referring to Adobe, their PDF Reader application has been downloaded over 500 million times worldwide [6]. In other words, over 500 million users are out-of-the-box able to view PDF documents containing 3D content, without installing any additional software or dealing with license costs.

Although 3D support in PDF files is available since 2005, 3D data in such documents is rare. This may be referrable to the absence of sufficient and at once affordable software tools, creating such documents for arbitrary fields of application. For example, the manufacturing industry is one of the few branches which has already adopted PDF as a suitable format for exchanging 3D models. Since sections and 2D drawings are often less than ideal and insufficient for describing free-form surfaces, almost all major CAD applications, just to mention Dassault Catia V5, Bentley MicroStation and Graphisoft ArchiCAD, are already able to export 3D PDF documents. In contrast to these commercial products, one of the first freely available, open-source tools for integrating 3D content in PDF documents, is the *movie15* LaTeX package from Alexander Grahn [25]. It provides a lot of PDF 3D-related adjustment possibilities but it is not well-suited for dynamic integration of 3D content enriched with related semantic metadata.

Since 3D data without any kind of additional information is pleasant but rather useless, most 3D objects are related to further metadata, describing the 3D content or its context more precisely. In some areas, like cultural heritage, 3D related information is essential. Hence, users benefit from both a 3D visualization and additional information related to the presented 3D content, like it can be done by using X3D embedded in a HTML web page. For example, Kadobayashi [34] proposed an automatic blogging system, where users can share annotations made to 3D content, enabling collaborative experiencing.

In contrast to the X3D/HTML example and the automatic blogging system, PDF documents can contain both, the 3D content and the related metadata within one file, which enables the user to save this file locally, archive it or send it to another interested person. Therefore, a flexible method for generating PDF documents that contain rich 3D content is desirable, but has apparently not yet received much research attention. In [12] Barnes and Fluke proposed the incorporation of 3D content in PDF documents as a substantial improvement for the visualization of multi-dimensional data in astronomy research papers. Since multi-dimensional data is no astronomical phenomenon, their ideas for a more appropriate knowledge transfer by using 3D visualizations is of course also applicable in numerous other fields. Nevertheless, their approach just uses 3D content in a quite static way, just for visualization purposes.

Strobl et al. [43] published a workflow for the generation of PDF documents that contain 3D objects, which are dynamically enriched with embedded annotations to provide additional information on some specific parts of the 3D content. Although the proposed workflow greatly shows how 3D content can on-the-fly be enriched to provide additional information, it has several drawbacks. The major limitation of this approach is that it is based on the commercial software Adobe Acrobat Extended.

Summarising, it can be stated that research in the field of 3D-supported documentation with a bidirectional linking between 3D content and related textual information is still in its infancy in theoretical, methodological and most of all practical terms.

To facilitate the generation of such 3D-contained PDF documents, the work described in this thesis is exclusively based on open standards and freely available open-source software. A general overview of this approach is given in Figure 1.1, showing the three major parts, the underlying PDF specification, the 3D content and arbitrary additional information encoded in a well-formed XML (*Extensible Markup Language*) [17] document. At this point it is to say that this work focuses on the successful integration of 3D content with related information in PDF documents and not on the generation of the 3D content itself. Therefore it is assumed that the used 3D objects are available in form of a U3D (*Universal 3D*) [20], or PRC (*Product Representation Compact*) [8] file.

Based on a well-formed XML intput, the two-stage XSL-FO (*Extensible Stylesheet Language - Formatting Objects*) [13] workflow is used to transform the given data into a PDF document.

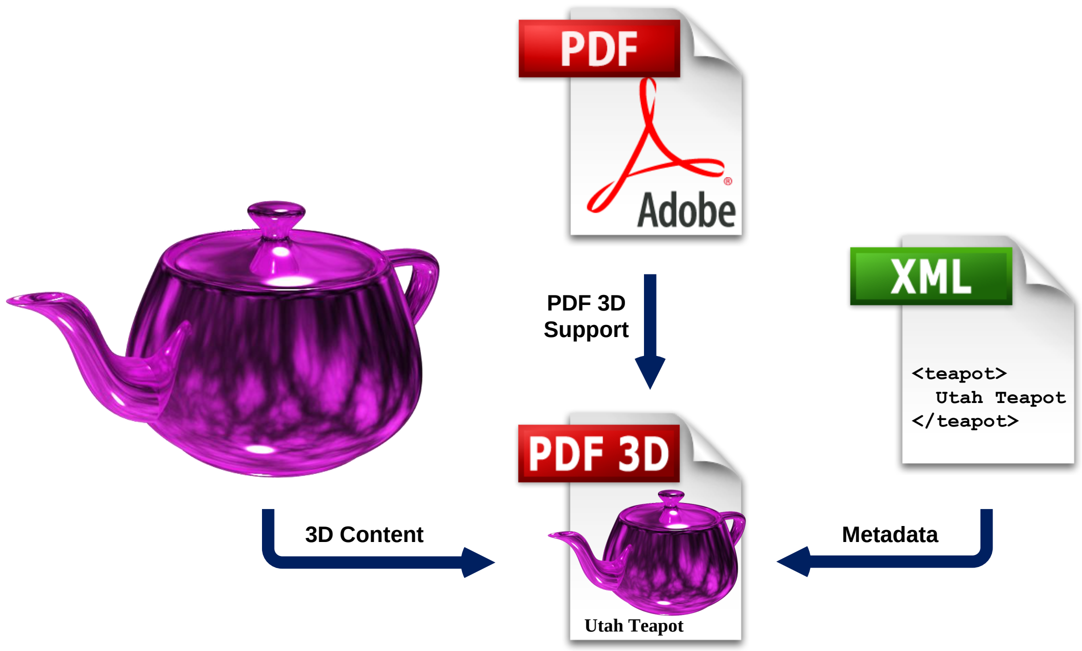


Figure 1.1: Schematic overview of the main parts used for the generation of a PDF document containing rich 3D content.

Since XSL-FO is traditionally two-dimensional and does not provide 3D support by default, the key objective of this work is to provide an extension for the integration of three-dimensional data, while preserving XSL-FO standard compliance. Therefore an XML vocabulary, referred to as FO3D, for representing 3D-object-related data within an FO document is proposed. Since FO processors are not by default able to process the proposed FO3D vocabulary, an extension has to be applied. Similar to the support for SVG (*Scalable Vector Graphics*) [33] included in an XSL-FO document, an FO processor can smoothly be extended to support the FO3D vocabulary too and this is shown by an exemplary practical implementation, based on the open-source FO processor Apache FOP [45].

Since XML and XSL-FO are approved standards for the automated generation of documents, the proposed approach can easily be integrated in existing documentation workflows, as well as be used in webservice environments. FO3D is a lean, flexible and extensible approach for integrating 3D content and related semantic meta-information of any kind in 3D PDF documents, which is shown in this thesis.

1.1 Outline

This work is organized as follows. Chapter 2 introduces basic technologies and fundamental concepts, on which the subsequent chapters rely. A more detailed description of available software tools and related research is given in Chapter 3, directly followed by the theoretical part of the proposed approach for integrating three-dimensional content and semantic metadata from multiple sources in a single PDF document. This detailed description is followed by Chapter 5, which outlines the application of knowledge by a practicle implementation based on the open-source XSL-FO processor Apache FOP. Afterwards, Chapter 6 presents three example scenarios and discusses the resulting PDF documents, containing interactive 3D content and various kinds of related meta information. Finally, Chapter 7 concludes this thesis with a discussion of achieved goals, as well as an outlook on future work.

Chapter 2

Basic Technology

This chapter briefly introduces fundamental concepts this work is built on. The main topics are the Portable Document Format (PDF), the Extensible Markup Language (XML) as well as some closely related specifications and an introduction to what is meant by the commonly used term "metadata".

2.1 Portable Document Format

2.1.1 Historical Overview

In the year 1991, with the idea of a new technique for viewing, printing and sharing information with other people, John Warnock, co-founder of Adobe Systems Incorporated, wrote a paper that he referred to as "The Camelot Paper" [53]. In this paper, Warnock outlined the most significant goals of this, at that time new approach as follows:

> *"This project's goal is to solve a fundamental problem that confronts today's companies. The problem is concerned with our ability to communicate visual material between different computer applications and systems."*

This paper denotes the origin of the nowadays well-known and massively used Portable Document Format. As previously mentioned, the fundamental ideas of PDF have been the exchange and presentation of electronic documents in an easy and reliable way, independent of the environment in which they are created, viewed or even printed. Warnock further outlined:

> *"Imagine being able to send full text and graphics documents (newspapers, magazine articles, technical manuals etc.) over electronic mail distribution networks. These documents could be viewed on any machine and any selected document could*

be printed locally. This capability would truly change the way information is managed."

From the early days until today, the PDF specification has been continuously improved and has recently reached the status of an ISO standard. The historical development of PDF is briefly outlined in the following [41].

PDF 1.0 and Acrobat: On June 15, 1993 Adobe officially released the specification of PDF version 1.0 and coincidently published an application for the creation and the viewing of such electronic documents, named Acrobat. Prior to the public product name "Acrobat", its code name has been "Camelot", which has later been replaced by "Carousel". PDF 1.0 already supported the embedding of fonts, internal links and bookmarks, but the only supported color space was RGB. In the beginning, Adobe tried to sell the Acrobat Reader for about $50, which was not conducive for the popularity of the format.

PDF 1.1 and Acrobat 2: With the release of Acrobat 2 in 1994, Adobe dropped the price and started to distribute the Acrobat Reader for free. The corresponding PDF version 1.1 added support for external links, article threads, device independent color, notes and security features. Apart from Adobe itself, the US tax authorities also early adopted PDF documents for the distribution of their tax forms. In Acrobat 2.1, for the first time, support for multimedia, especially audio and video data, was added. At this time, other portable and device independent formats, such as Envoy or DejaVu, also tried to claim market shares.

PDF 1.2 and Acrobat 3: In November 1996, Adobe published PDF 1.2 and the matching Acrobat version 3.0. This release added support for the *Open Prepress Interface* (OPI) 1.3 specifications, the CMYK color space, spot colors, halftone functions and overprint instructions. These enhancements brought the prepress industry to the scene. Support for JavaScript or more specifically JavaScript 1.2 was, for the first time, introduced in Adobe Acrobat Exchange 3.01 via the "Acrobat Forms Author Plugin 3.5 Update". More detailed information on JavaScript is presented in Section 2.1.3. With the release of a plugin for viewing PDF documents in the web browser Netscape Navigator, the popularity of PDF increased along with the booming Internet. Based on the PDF 1.2 specification, a consortium of prepress companies, due to reliability issues, released the PDF/X-1 standard in 1998. PDF/X will be described in the following.

PDF 1.3 and Acrobat 4: Acrobat 4 and the PDF 1.3 specification were launched in April 1999 and added support for OPI 2.0, smooth shading and annotations like free text, highlight, underline, strikeout, stamp, popup and file attachments, respectively. At this time, the Acrobat Reader has been downloaded from the Internet over 100 million times and the prepress industry had widely adopted PDF as a useful tool for file exchange, troubleshooting and/or softproofing.

PDF 1.4 and Acrobat 5: Over two years later in May 1999, Adobe released the next generation, Acrobat 5 and the PDF 1.4 specification. Numerous new features, like the support for transparency, improved security by 128-bit encryption, support for JavaScript 1.5 and "Tagged PDF", which facilitates the repurposing of the contained content by adding additional structural metadata. Acrobat 5, for example, significantly enhanced the forms-functionality, improved the support for the integration with Microsoft Office and added or at least improved export-filters, including the export from PDF to the *Rich Text Format* (RTF), facilitating the use with word processors.

PDF 1.5 and Acrobat 6: In April 2003, along with the specification of PDF 1.5, Adobe announced version 6 of the Acrobat software ensemble. In contrast to a few namable changes in the PDF specification, like improved support for compression techniques including JPEG 2000, layer support and some improvements concerning tagged PDF, the Acrobat software collection offers far more innovations. The first important change was that the Acrobat Reader and the Adobe eBook Reader merged and got renamed to "Adobe Reader". As a consequence, the file size of the PDF viewing application from Adobe has grown. The high-end version of Acrobat 6, called Acrobat Professional, for example, also enables integrated preflighting, PDF optimization, rulers and guides, PDF/X support, as well as loupe and measurement tools. Most of these newly added tools are again tailored to the use in prepress workflows, but can certainly also be used for other purposes.

PDF 1.6 and Acrobat 7: Acrobat 7 was released in January 2005 and offered support for PDF version 1.6, which enhanced the available range of functions, among others, by improved encryption algorithms, direct embedding of OpenType fonts and the ability to generate PDF files that act as a container for other PDF documents. Apart from these new features, the most impressive contribution to both, Acrobat 7 and the PDF 1.6 specification, is the ability to directly embed 3D content within PDF documents. Since this thesis is heavily based on the 3D support of PDF, a more detailed introduction to this feature is given in Section 2.1.4.

PDF 1.7 (ISO 32000-1) and Acrobat 8: With the release of PDF 1.7 in November 2006, Adobe added improved support for security, commenting as well as enhanced settings for the embedding of 3D content. Acrobat 8 was released in October 2006 and despite the fact that PDF 1.7 was supported, Acrobat 8 still used PDF 1.6 as its default version. Apart from improved support for PDF/A, better organized menus and toolbars, the preflighting tool has been enhanced to handle a number of corrections, called “fix-ups”. Adobe submitted the PDF 1.7 specification to the ISO and in 2008, PDF 1.7 became an offical ISO-standard, referred to as ISO 32000-1:2008 [4].

PDF 1.7 (ISO 32000-1) and Acrobat 9: Currently the most recent version, Acrobat 9, was published in June 2008, but since the ISO is controlling the PDF standard now, Adobe is not in the position to release a PDF 1.8 specification. The ISO is responsible for the development of new PDF standards. Therefore, Acrobat 9 still sticks to PDF 1.7 as basic version, but also supports custom extensions [7]. These custom extensibility features of PDF 1.7 are documented in ISO 32000-1:2008 in Annex E. For example, Acrobat 9.0 implements PDF 1.7 and Adobe Extension Level 3, as of Acrobat 9.1 Adobe Extension Level 5 is supported in addition to the inherited features from the underlying PDF 1.7 specification. Of course other companies, apart from Adobe, can also define extensions and/or request changes to extensions that have been proposed by Adobe. Since more companies and organizations are involved in the specification of new PDF standards now, it will take more time to specify the next version of PDF.

PDF 2.0 - ISO/NP 32000-2: The next version of the PDF specification, version 2.0, is currently under development (ISO/NP 32000-2). For now it has reached stage 10.99, which stands for “New project approved”.

Apart from the previously described main PDF standards, the ISO has additionally approved and developed the following substandards of PDF:

PDF/X is, as already mentioned, a substandard that is primarily used in the printing industry. Multiple PDF/X standards exist and each is based on a certain version of PDF, for example PDF/X-1 is based on PDF version 1.2. The PDF/X standards reduce the range of functions, available in the underlying PDF specification, to a certain set that is useful for the area of printing and graphic arts. Additionally, it contains a detailed description of how the structure of such a PDF document should look like to allow for automated interchange. For example, a valid PDF/X document includes all required fonts, respectively images in high resolution, and typically guarantee the same visual appearance on almost any kind of PostScript device. All PDF/X standards are controlled by ISO, referred to as ISO 15929 [29] and ISO 15930 [32].

PDF/E is a substandard that defines the creation of PDF documents for engineering workflows and has been published by the ISO as ISO 24517 [30]. It is based on PDF 1.6 and has been designed to be an open format for engineering and technical documentation.

PDF/A is designed as a substandard for archiving in corporate, government, library and many more environments. PDF/A-1 is based on PDF 1.4 [28]. The central goal of this standard is to ensure that documents created today can also be viewed many years in the future. Currently, the next version of PDF/A-2 (ISO/DIS 19005-2) is under development and will be based on PDF 1.7, including new features such as the archiving of PDF documents with embedded 3D content.

Referring to the last sentence in the second citation, taken from the "Camelot paper", and with respect to the development in the last two decades, PDF really changed the way information is managed in a quite sustainable way. To name but a few outstanding facts, over 500 million copies of the Adobe/Acrobat Reader software have been downloaded until today, more than 200 million PDF documents are currently posted on the Internet and the U.S. *Internal Revenue Service* (IRS) has recorded over 1.5 billion PDF tax form downloads since 1996 [6].

2.1.2 PDF Internals

Based on the advanced imaging model derived from the page description language PostScript also published by Adobe back in the year 1983, PDF is enabled to describe graphics and text in a device and resolution independent way. In contrast to PostScript, which is a programming language, PDF defines a more structured, strictly page-oriented, binary file format that is optimized for high performance in interactive viewing. In addition to the standard content stream of a PDF document, annotations like hypertext links and comments allow for interactive use and collaboration. Annotations are not part of the page content itself, which provides a well defined separation of real content and additional information.

A schematic overview of the PDF document structure, which can be seen as a hierarchical tree of objects, is shown in Figure 2.1. Most of the objects that are contained in a PDF document are "dictionaries" which will be described in the following. The topmost element of PDF documents, the root node, is represented by the *Catalog Dictionary*, which references the content of the document, arranged in a sequence of page objects. Additionally, outlines, article threads, named destinations and many other attributes can be defined too. Apart from content-related information, the catalog dictionary also contains information about how the document shall be displayed on the screen. The Portable Document Format defines nine basic types of objects, which can also be labeled, so that they can be referred to by other objects. A brief introduction to the basic types of PDF is presented in the following enumeration:

1. **Boolean** objects represent the two logical states true and false with the equally named keywords "true" and "false".

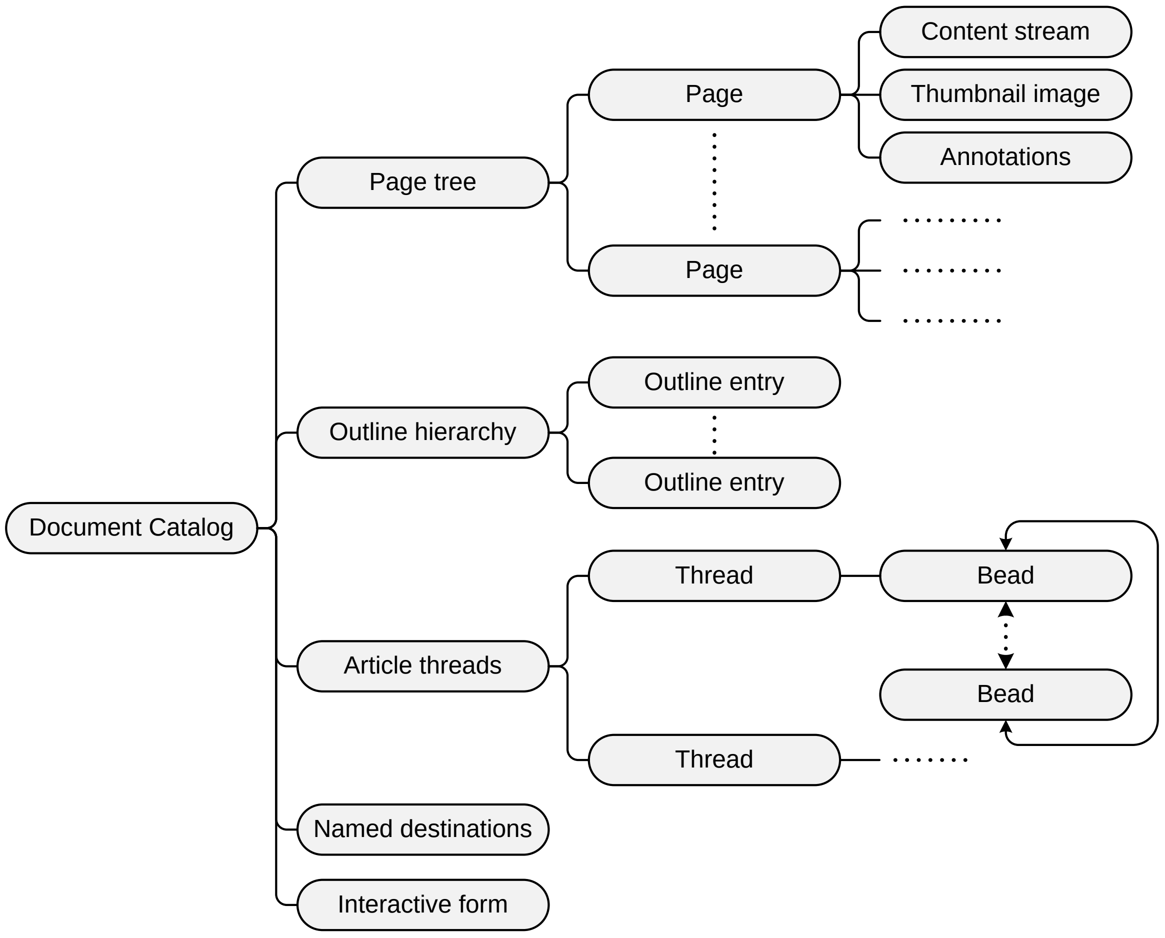


Figure 2.1: Schematic structure of a PDF document [4].

2. **Integer** objects, as the name suggests, represent mathematical integers written as one or more decimal digits optionally preceded by a sign.
   Examples: `390`, `0`, `-3923`, `+29`

3. **Real** objects represent mathematical real numbers written as one or more decimal digits with an optional sign and a leading, trailing, or embedded period (decimal point).
   Examples: `0.83`, `.329`, `+33.`, `-392.39`

4. **Strings** typically consist of zero or more bytes and can be written in two different ways. Literal strings are written as a sequence of literal characters enclosed in parentheses and hexadecimal strings represent hexadecimal encoded data enclosed in angle brackets.
   Examples: `(Literal string)`, `<4E6F762073686D6F7A206B6120706F702E>`

5. **Name** objects have been available since PDF 1.2 and represent atomic symbols uniquely defined by a sequence of any characters except null, starting with a solidus.
   Examples: `/Type`, `/Fullscreen`, `/Rect`, `/OnInstantiate`

6. **Arrays** are one-dimensional object collections that have zero or more entries, which are arranged sequentially. In contrast to arrays known from other computer languages, PDF Arrays may contain heterogeneous elements (objects of different type), including even other arrays. PDF arrays are written as a sequence of objects, separated with a blank character and enclosed by square brackets.
   Example: `[true 123 0.382 (String) /Name [...]]`

7. **Dictionary** objects are unordered, associative tables containing pairs of keys and their corresponding values, but multiple entries may not have the same key. In difference to PostScript, where dictionary keys may be objects of any type, keys of PDF dictionaries always have the previously described type *Name*. However, the value of a dictionary entry can be an object of any type available in PDF documents. PDF dictionaries are written as a sequence of key-value pairs enclosed in double angle brackets.
   Example: `<</Num 0.87 /Text (a string) /Dict <<...>> ...  >>`

8. **Streams** are, similar to string objects, sequences of bytes, with the minor difference that stream objects may be of unlimited length, at least in theory. Therefore, objects with potentially large amounts of data, for example images or 3D models, are represented as streams. A stream object consists of a dictionary, directly followed by zero or more bytes surrounded by the keywords "stream" (followed by a newline) and "endstream". The PDF specification claims that all stream objects must be encoded as "indirect objects", which will be described later. The following listing is an example of how stream objects are encoded in PDF documents.

```
<< ... dictionary ... >>
stream
... zero or more bytes ...
endstream
```

9. Finally, the **null** object, denoted by the keyword “null”, has a type and a value different to any other of the previously mentioned objects. For example, specifying `null` as the value of a dictionary entry is equivalent to omitting the entry entirely.

The previously mentioned labeled objects are more correctly called “Indirect Objects”, which other objects can refer to by a unique object identifier. Indirect objects are labeled with a positive integer designating the object number, followed by a non-negative number representing the generation (version) number, followed by the keyword “obj” and having “endobj” after it. The object number can be assigned sequentially within a PDF document, but may also be applied in any arbitrary order. The generation numbers of indirect objects within a newly created document are all set to zero, but may be incremented when the document is updated. The combination of object and generation number provides a unique identifier by which indirect objects can be referred to.

To clarify the described mechanism, Listing 2.1 contains a simple example of two indirect objects and their conjunction. It shows a literal string object and a dictionary object, having only one entry named “Title”. The value of the “Title” entry is not assigned directly, but referred to by using an indirect object reference. Such indirect references are established by making use of object and generation number in the form of “`<o> <v> R`”, where `<o>` is the indirect object number, `<v>` is its version number and `R` is just the uppercase letter R as a hint for the reference itself.

```
7 0 obj
(This is an indirect literal string object.)
endobj
...
31 0 obj
<< /Title 7 0 R /Keywords (direct literal string object) >>
endobj
```

Listing 2.1: Example of two indirect objects and an indirect object reference.

All indirect objects are listed in a table, called *Cross Reference Table*, that can be found at the end of a PDF file. The cross reference table is a data structure, representing the byte offset to the beginning of every indirect object contained in a PDF file. This feature allows for random access, without the need of parsing the complete PDF file before.

2.1.3 JavaScript Support

JavaScript is an object-oriented scripting language, primarily used in the form of client-side JavaScript that is implemented as an integrated component of web browsers. It is a dialect of ECMAScript [22] and was first-ever introduced by the web browser software Netscape Navigator in 1995. Due to the similarity in names, JavaScript is often confused with Java, but despite the same name it is essentially unrelated to the Java programming language. Those two languages

Adobe Acrobat Version	3.01	4.0	5.0	6.0	7.0	8.0	9.0
JavaScript Version	1.2	1.2	1.5	1.5	1.5	1.6	1.7

Table 2.1: Different versions of Adobe Acrobat and the correspondingly supported version of JavaScript.

just share several concepts, for example both are object-oriented, have a C-like syntax and are normally used in a “sandboxed” environment, which means that the program code is executed in an isolated environment typically just having access to a tightly-controlled set of resources.

By means of JavaScript, a developer can enhance both, the range of functions available in a PDF document or even in the PDF viewer application itself. Due to the lack of PDF viewers that comprehensively support the JavaScript specification of Adobe, the following explanations will refer to Adobe Acrobat or the Adobe Reader, respectively.

Adobe Acrobat first-ever introduced support for JavaScript in version 3.01. In Table 2.1, a more detailed overview of Acrobat versions and the correspondingly supported version of JavaScript is given [3]. The JavaScript implementation of Acrobat is an extension of the core JavaScript, including objects like *Math*, *String*, *Date*, *Array* or *RegExp*, and implements additional features in the form of new objects and their accompanying methods and properties. Generally, Acrobat and other PDF viewers distinguish between two different types of PDF related JavaScript:

Document Level JavaScript is a broader term for JavaScript code that is directly embedded in a PDF document and can for example be useful for tasks like validating form fields, time-controlled animations, user interaction, etc. and is strongly related to the document and its components.

Folder Level JavaScript is in contrast to the previously mentioned type more closely related to the PDF viewer application and can, for example, be used for extending the viewers JavaScript API with additional plugins or adding menu buttons. Referring to Adobe Acrobat, such folder level scripts can be added to the viewer by placing JavaScript files in special folders. Acrobat looks for and loads these scripts on startup from two system dependent locations in the file system, which can be identified by executing the code shown in Listing 2.2 from the Acrobat JavaScript console window.

```
// global folder for document level JavaScript
app.getPath("app","javascript");

// user dependent folder for document level JavaScript
app.getPath("user","javascript");
```

Listing 2.2: Retrieving file system locations for storing folder level JavaScript in Adobe Acrobat.

Throughout the Acrobat JavaScript API, functions may be restricted for security reasons. For example, folder level scripts are granted with higher authorization than document level scripts. This means that some functions may not work in JavaScript code that is embedded in a PDF document.

JavaScript enables PDF creators to build rich documents communicating with a database, handling file attachments, managing document security, embedding interactive forms and so on. Beginning with Acrobat 7, there have been extensive improvements to the JavaScript functionality and today (Acrobat 9) JavaScript is an integral part of PDF documents.

2.1.4 Support for 3D Content

Since version 1.6 (Acrobat 7), PDF documents have been able to embed three-dimensional objects also referred to as 3D artwork, such as those used by CAD software. Multiple instances of 3D artwork can be embedded within rectangular areas on the pages of a PDF document. Those areas can be overlaid with two-dimensional content that is, for example, used if the 3D artwork is disabled or for high-resolution printing. In difference to two-dimensional images, instances of 3D artwork are not referenced in the main content stream of a page, but attached as an annotation, outlined in Figure 2.1.

Apart from the three-dimensional content itself, a 3D artwork can have multiple predefined views including a default view that is displayed initially. Views may also have a title that can be presented and selected in a user interface, to switch between different appearances of the 3D scene.

A fundamental part of interactive 3D artwork is JavaScript that is used to manipulate 3D objects and their appearance programmatically. The JavaScript code, attached to a 3D artwork, is executed on the initialization of the associated 3D annotation. Referring to the Acrobat PDF viewer, JavaScript that is used in 3D annotations is running in a separate context having its own API specification [2]. The 3D JavaScript API provides lots of useful objects for developing 3D-related applications, for example *Vector3*, *Matrix4x4*, *Quaternion*, *Color* and *Camera*. To make the 3D scene aware of user interaction, several event handlers such as the *KeyEventHandler*, *MouseEventHandler*, *SelectionEventHandler* or *ScrollWheelEventHandler* can be utilized.

The standard Acrobat JavaScript API, respectively the 3D JavaScript API, is continuously being improved and strongly depends on the PDF viewer application. For example objects like *Light*, *Node* or *Scene* have not been present in the 3D JavaScript API of Acrobat 7, but they have been available since Acrobat 8.1. Therefore, the dedicated PDF version and the features available in the chosen PDF viewer are important parameters for developing 3D visualizations embedded in PDF documents.

The 3D content itself is embedded in a PDF document as a stream object, which is linked to the 3D annotation by using an indirect object reference. Today, the 3D stream can contain two different formats representing 3D content and both will be introduced below.

Universal 3D File Format (U3D)

Back in the year 2003, a special consortium called *3D Industry Forum* (3DIF) started to work on an open and extensible file format to facilitate the reuse, the sharing and the visualization of 3D data in any mainstream application. Some well-known members of the 3DIF are for example Adobe Systems Inc, ATI, Bentley Systems, The Boeing Company, Cinema4D, Discreet, Fraunhofer Institute, Hewlett-Packard, Intel Corporation, Lego, Right Hemisphere and SolidWork [21].

Since 2004, the outcome of this work has been associated with Ecma International, which formed the *Technical Committee 43* (TC43) to specify the U3D file format primarily intended for downstream 3D CAD re-purposing and visualization purposes. In December 2004, the General Assembly adopted the 1st Edition of the U3D file format that was standardized by Ecma International in August 2005 as ECMA-363 [20]. In the 2nd Edition of U3D, the ability to extend the file format was added by the TC43 and this release was submitted to ISO/IEC for fast-track processing. With the 3rd Edition, in the year 2006, several changes, received as comments during the ISO/IEC ballot process, were incorporated. The 4th Edition was released in 2007 and adds support for uniform and non-uniform, rational and non-rational free-form curves and surfaces to the U3D file format. Some of the core features of U3D are listed below:

- Binary encoding
- Domain-specific compression
- Continuous level of detail
- Progressive data representation
- Animation support
- Extensibility

To accelerate the distribution and facilitate the usage of U3D, the 3DIF started to work on a library written in C++, which is today freely available as an open-source project[1].

U3D was chosen as the format for embedding 3D objects into PDF documents. Support for the 3rd Edition of U3D was added in Acrobat 8.1, which also started the support of the PRC file format described in the following section.

[1]`http://u3d.sourceforge.net`

PRC File Format

In addition to the open U3D file format, Acrobat also supports the PRC format, which is developed by Adobe and stands for *Product Representation Compact* [8]. Acrobat has been supporting the PRC format since version 8.1 (PDF 1.7 Extension Level 1). PRC is a sequential binary file format, optimized to store, load and display various kinds of 3D content. With a special focus on *Computer Aided Design* (CAD), the following concepts are supported by this format:

- Assemblies and parts
- Hierarchical trees of entities (Coordinate systems, Wireframes, Surfaces, and Solids)
- Exact representation of geometry
- Tessellated (triangulated) representation
- Markup

Although the PRC file format was developed by Adobe, its specification is available for free and the first third-party tools are in the making. For example, the vector graphics language *Asymptote*[2], a natural coordinate-based framework for technical drawing, produces PDF documents containing labeled 3D-plots, by generating 3D data that is encoded in the PRC file format. To accelerate innovation and achieve broader adoption of PRC, Adobe in December 2008 released control for the PRC specification to the ISO, which is working on the standardization as an extension of the ISO 32000 standard effort.

2.2 XML Related Standards

The Extensible Markup Language (XML) is a simple, but very flexible text-based format and also a fee-free open W3C standard [17]. It plays an increasingly important role in interchanging structured textual data between computer programs. Another important part of its success is that it is not only readable and writable by computer programs, but also by humans, using nothing else but a text editor. The origin of XML can be found in the Standard Generalized Markup Language (SGML), which is ISO-standardized under ISO 8879 [26]. XML itself is a profile, a subset of SGML, designed to ease the implementation of parsers compared to full SGML parsers. Most XML qualities are already defined by SGML, to name but a few:

- Separation of logical and physical structures (elements and entities)

[2] `http://asymptote.sourceforge.net`

- Availability of grammar-based validation
- Separation of data and metadata (elements and attributes)
- Mixed content
- Separation of processing from representation (no processing instructions)
- Default angle-bracket syntax

The initial XML draft was published in November 1996 and the first W3C Recommendation of XML 1.0 was released on February 10, 1998. Since its first release, the XML 1.0 specification has been continuously revised and reached the most recent 5th Edition.

A simple example of an XML document, containing exemplary information about a person, is presented in Listing 2.3.

```xml
<?xml version="1.0"?>
<person id="1234">
  <firstname>John</firstname>
  <lastname>Doe</lastname>
  <email>john.doe@example.com</email>
</person>
```

Listing 2.3: Example of a valid XML document

Until today, numerous closely XML-related specifications and XML dialects have been developed. A few selected and interesting ones, especially regarding the topic of this thesis, will be described in greater detail.

2.2.1 XML Namespaces

The use of XML Namespaces enables a single XML document to contain elements and attributes taken from different XML vocabularies, without having any naming conflicts [16]. The further qualification of elements and attributes is done by associating them with namespaces identified by URI (*Uniform Resource Identifier*) references [15]. This practice removes the ambiguity of elements and attributes that have the same name, but belong to different vocabularies and therefore have an unequal meaning. A wide-spread technology that massively uses XML Namespaces is the Extensible Stylesheet Language described in Section 2.2.2.

```xml
<?xml version="1.0"?>
<order xmlns="urn:ex:order" c:xmlns="urn:ex:customer" p:xmlns="http://prod.com/ns">
  <id>9872</id>
  <c:id>39202</c:id>
  <p:id>4832</p:id>
</order>
```

Listing 2.4: Exemplary usage of XML Namespaces

An exemplary usage of XML Namespaces is presented in Listing 2.4. This example shows an XML document that represents an order, containing information about a customer and the ordered product, both identified by an element named "id". Without the use of XML namespaces this would end up in an ambiguous state, because there is no additional information that tells the `id` elements apart. The most simple and naive solution to solve such ambiguities would be the renaming of equally named elements, for example to: "customer-id" and "product-id", but this is often impossible when different externally defined XML vocabularies should be combined within one document. In other cases it is possible to (re)define an XML structure, but often things become more complicated than when element and attribute names as they are, just associating them with separate namespaces.

2.2.2 Extensible Stylesheet Language

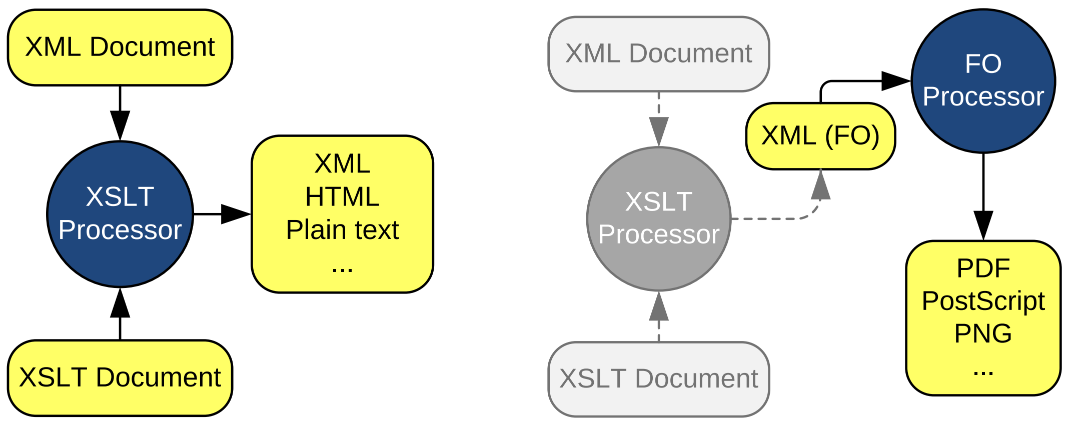
The *Extensible Stylesheet Language* (XSL) belongs to the family of XML standards, which are mostly developed by the W3C, in this case especially by the W3C XSL Working Group. Similar to XML itself, which is strongly related to the SGML ISO standard, the XSL specification was inspired by the *Document Style Semantics and Specification Language* (DSSSL), which is also ISO-standardized and was designed for the transformation and presentation of SGML documents [27]. During the development process of XSL, the initially planned XML-equivalent to DSSSL was split up into a two-stage process, applying a structural transformation first, directly followed by the formatting procedure. Today the XSL family consists of three separate specifications, each covering its own special field. The three major subsets of XSL are listed below and will be described later:

- XML Path Language (XPath)
- XSL Transformations (XSLT)
- XSL Formatting Objects (XSL-FO)

Because of much confusion around the abbreviation XSL, applied in so many different meanings, Michael Kay in [36, page 22] suggests the avoidance of this term and proposes the use of the proper name XSL-FO instead. When using the term XSL one may mean XSL-FO, but others may think of XSLT. The three subsets of the XSL specification can be used in combination but also independently from each other and will be explained particularly in the following.

XML Path Language (XPath)

XPath is a non-XML language used by XSLT but can also be utilized in a non-XSLT context [14]. The language is designed to address parts of an XML document, which includes elements as well



(a) XSLT workflow diagram **(b)** XSL-FO workflow diagram

Figure 2.2: Workflow diagrams of the XSLT process (a) and the XSL-FO process (b), which may contain XSLT as input stage.

as attributes and is therefore well-suited for XML-tree navigation purposes. XPath 1.0 became a W3C Recommendation on November 16, 1999. Its successor XPath 2.0, which is currently the most recent version of this language, became a W3C Recommendation on January 23, 2007. Even though version 2.0 provides a greatly expanded list of new features and more flexibility in dealing with data types, version 1.0 today is still more widely used, which may be referable to the much larger language specification and some fundamental changes in basic concepts.

XSL Transformation (XSLT)

XSLT is, according to the very first sentence in its specification, a language for transforming XML documents into other XML documents [35]. However, XSLT is also capable of transforming XML documents to other non-XML formats like HTML or text-based formats. Therefore, Michael Kay, member of the W3C XSL Working Group and author of the Saxon XSLT processor, proposes in [36] a minor redefinition in the form of:

"XSLT is a language for transforming the structure
and content of an XML document."

The declarative XML-based markup language XSLT was originally intended to perform complex styling operations, like for example the automatic generation of indexes, but nowadays it is used as a more general purpose XML processing language that is also widely used for other purposes beyond styling and formatting. As denoted before, the transformation and visual presentation of XML documents has been separated during the design process of XSL.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3    <xsl:output method="text"/>
4    <xsl:template match="person">
5      <xsl:value-of select="concat('Person ',@id,': ')"/>
6      <xsl:value-of select="concat(firstname,' ',lastname)"/>
7      <xsl:text> (</xsl:text>
8      <xsl:value-of select="email"/>
9      <xsl:text>)</xsl:text>
10   </xsl:template>
11 </xsl:stylesheet>
```

Listing 2.5: Example of an XSLT document including XPath

Figure 2.2a provides a simple graphical overview of the XSLT processing workflow, which is for example used to transform the XML document, already introduced in Listing 2.3, into the plain text output shown in Listing 2.6. The applied XSL transformation itself is shown in Listing 2.5 and also includes XPath statements, like for example, the `concat` function in line 5 or the `email` selector in line 8.

```
Person 1234: John Doe (john.doe@example.com)
```

Listing 2.6: XSL Transformation result of applying the transformation in Listing 2.5 to the XML document shown in Listing 2.3.

XSL Formatting Objects (XSL-FO)

XSL-FO is an XML-based markup language (an XML vocabulary) designed for the formatting and presentation of XML documents [13]. It defines a set of elements and attributes having a special meaning to describe the visual appearance of XML-encoded documents and is therefore well-suited to separate data from its presentation. Many attributes in XSL-FO are, because of their analogous meaning similar to style-properties known from another widely accepted W3C Recommendation, referred to as Cascading Style Sheets (CSS) [52], which are primarily used in web pages and browsers.

The general idea behind XSL-FO is that any data stored in an XML document can be transformed to another XML document containing both, the data and the XSL-FO elements describing its visual presentation. This transformation is done by making use of XSLT and XPath, denoted before. Once an XSL-FO document is available, an application called FO processor (FOP) is able to convert it into something that is either readable or printable or even both. Most often XSL-FO is used to generate PDF documents, but it is designed to be an output-independent formatting system, supporting various render targets depending on the implementation of the FO processor.

A structural overview of the XSL-FO processing workflow is presented in Figure 2.2b. The XSLT process, also included in this illustration, is not obligatory for working with Formatting

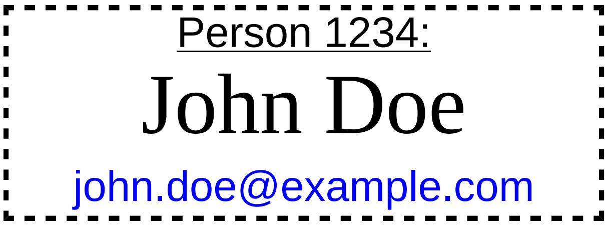


Figure 2.3: Visual typesetting result of the XSL-FO transformation shown in Listing 2.7 and the XML source document presented in Listing 2.3 (rendered by Apache FOP).

Objects, but most often it is used in combination to benefit from its advantages in transforming XML documents into XSL-FO documents. As denoted before, XSLT was originally designed for this purpose only, but nowadays it is utilized in a more general way. The misbelieve that XSLT is equal to the transformation of XML documents into XSL-FO documents is still widely spread, therefore many XSL-FO tutorials are tightly related to XSLT.

Currently the most recent version of XSL-FO is specified in the XSL 1.1 W3C Recommendation [13] released on December 5, 2006, which supersedes version 1.0 published on October 15, 2001. Referring to the XSLT and XPath language, which have already been available in version 2.0, a new XSL-FO version 2.0 is currently under development.

Similar to the very popular and powerful automated typesetting system TeX [38], developed by Donald E. Knuth at the end of the 1970s, XSL-FO also belongs to this class of typesetting systems. In difference to an interactive typesetting system, automated systems are specifically tailored to the (automatic) processing of large data volumes, rendering on demand and to achieve consistent layouts according to its formatting rules. As the term ”automatic” suggests, those tasks will be performed without the need of human intervention.

To give an illustrative example of what XSL-FO is able to do, Listing 2.7 shows a transformation tailored to the XML document structure introduced in Listing 2.3. The visual result of this automated typesetting is shown in Figure 2.3.

2.3 Metadata

The Latin or English usage of the term “meta” is used to describe something transcendental, or beyond nature, but referring to the Oxford English Dictionary[3], “meta” originally came from Greek and denotes “with, across, or after”. Metadata, then, can be thought of as “data about data”. The most recent use of the term metadata is associated with the computer age and stands for descriptive information about (web) resources or documents. With different kinds of metadata, objects of interest such as books, digital images or electronic documents can be provided with additional information that is not directly included in the content of the objective

[3]http://www.askoxford.com

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:
    fo="http://www.w3.org/1999/XSL/Format">
  <xsl:output method="xml"/>

  <xsl:template match="/">
    <fo:root>
      <fo:layout-master-set>
        <fo:simple-page-master master-name="s" page-width="8cm" page-height="31mm">
          <fo:region-body margin="1mm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-reference="s">
        <fo:flow flow-name="xsl-region-body">
          <xsl:apply-templates/>
        </fo:flow>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>

  <xsl:template match="person">
    <fo:block font-size="16px" border="2px dashed #000" text-align="center">
      <fo:block text-decoration="underline">
        <xsl:value-of select="concat('Person ', @id, ': ')"/>
      </fo:block>
      <fo:block font-size="32pt" font-family="Times">
        <xsl:value-of select="concat(firstname, ' ', lastname)"/>
      </fo:block>
      <xsl:apply-templates select="email"/>
    </fo:block>
  </xsl:template>

  <xsl:template match="email">
    <fo:block color="#00F">
      <fo:basic-link external-destination="{concat('mailto:', .)}">
        <xsl:value-of select="."/>
      </fo:basic-link>
    </fo:block>
  </xsl:template>

</xsl:stylesheet>
```

Listing 2.7: XSL-FO example using XSLT and XPath

itself. Often the separation of basic content and metadata is a floating gray area, depending on the underlying workflow.

Wide-spread types of metadata are, for example:

- Title, keywords, (short) description, author and creation date of a book or document
- Camera settings and the geographical position where the photo has been taken
- An additional textual annotation for a particular part of a 3D object
- etc.

Numerous different metadata schemes are being developed across many disciplines, such as library science, arts, education, archiving and so on. With respect to the large variety, the following sections provide just a brief introduction to a few selected metadata schemes and their fields of application.

2.3.1 Dublin Core

The *Dublin Core* (DC) metadata set of elements is a standard in the fields of library and computer science [19]. It is used for describing a wide range of network resources, such as digital materials (video, sound, image, text) or composite media like web pages, in a simple yet effective way. An international, cross-disciplinary group of professionals from librarianship, computer science, text encoding, museums and other related fields of scholarship and practice established the semantics of DC, arranged in two levels, "Simple" and "Qualified". The Simple DC comprises 15 elements and the Qualified DC defines additional elements, as well as a group of element refinements (qualifiers) useful for resource discovery. The fifteen-element DC specification achieved wide dissemination and has been ratified as IETF RFC 5013 [40], ANSI/NISO standard Z39.85 [11], and ISO standard 15836 [31]. Similar to other reference models, the concepts and semantics of Dublin Core are designed to be independent of the syntax that is used to express the metadata. Therefore, it can be used in a variety of different contexts, as long as the metadata is in a suitable form for interpretation by both, human beings and automated systems.

2.3.2 ICOM-CIDOC CRM

The *International Council of Museums* (ICOM) is a non-governmental, non-profit, international organization of museums and museum experts, comprising numerous international committees. One of these committees is the *International Committee for Documentation* (CIDOC) that provides curators, librarians and information specialists interested in documentation, registration, collections management and computerization with the opportunity to collaborate. This committee further defined a Conceptual Reference Model, known as the "CIDOC CRM", providing definitions and a formal, extensible ontology for describing concepts and relationships used in cultural heritage documentation. Since September 9, 2006 the CIDOC CRM has become an official international standard referred to as ISO 21127 [18]. Currently the latest CRM version is 5.0.2, published in January 2010 and providing 90 entity classes such as: *event*, *person*, *site*, *image*, *date*, *material*. It also defines 148 properties to express relationships between these entities, for example: *is type of*, *consists of*, *was influenced by*, *took place at*, etc.

Referring to its specification, this Conceptual Reference Model is an ontology in the sense used in computer science and has been expressed as an object-oriented semantic model to satisfy the requirements of both, the documentation experts as well as the information scientists. It is designed to be implemented in any relational or object-oriented schema, including machine-readable formats such as RDF Schema, KIF, DAML+OIL, OWL, STEP, etc. and instances of the CIDOC-CRM can be encoded in RDF, XML, DAML+OIL, OWL or other formats.

2.3.3 Text Encoding Initiative

Since 2000, the *Text Encoding Initiative* (TEI), founded in 1987 and known as the TEI consortium, has been a consortium that is hosted by academic institutions in the US respectively in Europe and collectively develops a standard for the representation of texts in digital form [47]. The main achievement of the TEI is a set of guidelines, specifying encoding methods for machine-readable textual documents, which are mainly used in the humanities, social sciences and linguistics. Today, TEI is the de facto standard for the encoding of electronic texts in the humanities academic community. To describe the structural, renditional and conceptual features of texts, the TEI Guidelines define and document a markup language expressed as a modular and extensible XML schema. Due to its modularity it can be customized for particular production or research environments. A popular customization example is *TEI Lite*, which is a manageable subset of the extensive full TEI specification. Referring to the specification of TEI Lite, it was designed to meet 90% of the needs of 90% of the TEI user community and because of its simplicity TEI Lite has been widely adopted, particularly by beginners. Currently the most recent version of the TEI Guidelines have been published in November 2007, referred to as *P5*, where the "P" stands for "proposal".

2.3.4 Extensible Metadata Platform

The *Extensible Metadata Platform* (XMP), published by Adobe back in 2001, is a standard for embedding file-content related information, usually referred to as metadata, into the file itself [1]. Based on the W3C standard Resource Description Framework (RDF) [51], XMP is an open-source, labeling technology that can be extended to accommodate existing metadata schemas and therefore systems working with metadata information do not need to be rebuilt from scratch. Not only a whole document but also fine-grained elements included in a resource can be described by making use of XMP encoded metadata properties. Sets of metadata property definitions, relevant to a wide range of applications from a variety of vendors, are predefined for the use with XMP. Dublin Core, for example, is one of those predefined element sets. Serialized XMP properties are wrapped in packets and stored in a natural manner, depending on the underlying file format. The guidelines for the embedding of XMP packets in numerous file formats, such as TIFF, JPEG, JPEG 2000, GIF, PNG, HTML, PDF, SVG/XML, PostScript and EPS are also defined within the XMP specification itself. A growing number of third-party applications already support XMP. As denoted previously, in XMP, metadata consists of a set of properties, which are always associated with a particular entity referred to as *resource*. Each property has both, a *name* and a *value*, and together with a resource this triple defines a statement in the form of:

"The <*property-name*> of <*resource*> is <*property-value*>."

For example:

"The *author* of *this thesis* is *Gerald Buchgraber*."

Chapter 3

Related Work

Based on the work previously described in Chapter 2, various applications have already been developed and research has been conducted. This chapter provides additional information on related approaches and already existing software tools for generating PDF documents that contain interactive 3D illustrations.

Altough PDF has been well-suited to incorporate 3D content since 2005, this type of knowledge transfer has apparently not received much research attention and is rarely used in scientific publications. In [12] Barnes and Fluke proposed the use of 3D content in scientific astronomy research papers, because most data collections used by astronomers are intrinsically multi-dimensional. Nevertheless, they claimed that in contrast most research publications exclusively present data in a two-dimensional way. Furthermore, they complemented their S2PLOT programming library by a PDF-3D export, which is based on the commercial software tool Adobe Acrobat. They also noted that a freely available alternative would be highly desirable. In addition to the embedding of 3D content, they also showed low-level JavaScript interaction with the 3D visualization, but have not used additional resources, which can also be added to 3D objects and further improve the knowledge transfer.

An adequate representation of multi-dimensional data is not only relevant in the field of astronomy or astrophysics, but also in numerous other discipines. In [39] Kumar et al. stated that it is often crucial for an in-depth understanding of how multi-dimensional research data is presented in scientific publications. Especially in the fields of structural biology, chemistry, physics and medicine, where molecular features have been discussed over years by means of static or “stereo” images, they propose the use of 3D-contained PDF documents for a much better knowledge transfer. Altough various web-based molecule visualization tools are already available, the authors chose PDF because of its broad user acceptance, which made PDF the universal standard for electronic publications. In contrast to web-based approaches, requiring additional application or plugin installations, a PDF document has three major advantages, (i) a conforming PDF viewing application, like Adobe Reader, is most often already installed on the target system, (ii) a PDF document can also be viewed when no Internet connection is established, and (iii) essentially related information about the molecule can directly be supplied within the same document. Similar to Barnes and Fluke [12], the PDF generation workflow described by Kumar et al. is also based on the non-free commercial software tool Adobe Acrobat. Furthermore, Kumar et al. have also not exploited the full range of available capabilities, because additional resources combined with appropriate JavaScript code can be used to provide a better understanding of the displayed 3D content, for example, by adding a bidirectional link between 3D content and the surrounding information contained in the PDF document itself.

The previously described approaches just emphasize the benefit of using 3D visualizations in scientific publications, but do neither provide a non-commercial solution for the generation of such documents, nor do they contain methods for connecting 3D objects with related metadata. An example of how 3D content can be combined with related meta information has been presented by Kadobayashi in [34]. In this work, he described a web-based approach for interactively annotating 3D content while "walking" through the 3D scene. The underlying system then automatically creates a web page containing the annotations in "blog" format, referring to the specific spot where the annotation has been created.

Based on the work described in this thesis, the 3D blogging system could be enriched by an automated export, generating a PDF file that contains the 3D content and the bidirectionally interlinked textual annotations, within a single document.

Exactly the same feature has already been suggested by Strobl et al. [43], but their PDF generation workflow suffers from a few major drawbacks. Their PDF generation process is also based on the commercial software Adobe Acrobat Extended and the corresponding Acrobat SDK (*Software Development Kit*), which makes it, due to license issues, nearly impossible to use this approach in a web service or server solution. Furthermore, they use the SDK for a direct manipulation of the 3D content itself and introduce additional geometry for every URL (*Uniform Resource Locator*) reference. Their approach just supports the PRC file format and not the clearly preferable U3D file format, which is an open standard independent from Adobe. As stated by Strobl et al., a generation workflow for 3D-contained PDF documents that is independant of commercial tools like Adobe Acrobat would be highly desirable. The work presented in this thesis proposes a new approach that is exclusively based on open standards and open-source software for filling just that gap.

The following sections provide an overview of related tools for generating PDF documents that can contain 3D objects. This does not include Adobe Acrobat because this de facto standard software for working with PDF documents has already been introduced with the historical development of the PDF standard in Section 2.1.1.

3.1 Ecrion XF Rendering Server

One of the first approaches for combining an XSL-FO document with simple 3D objects, is the XF Rendering Server from Ecrion [23]. On May 16, 2009 Ecrion Software Inc. published in a press release, in which they proclaimed that their XML formatter, XF Rendering Server 3.0, is able to embed 3D objects in PDF documents. Currently, the most recent version is called *XF Rendering Server 2010* and according to its documentation, 3D support is exclusively present in the full-featured *Ultrascale Edition*. The XML element, which enables the integration of 3D content, is named `object-3d` and is defined in a separate XML namespace, specifically created for proprietary extensions from Ecrion to extend the XSL-FO 1.1 specification. Apart from the 3D content which has to be available in form of an external U3D file, the lighting scheme and a two-dimensional graphics used if the 3D artwork is deactivated, can be defined too. Referring to a press release from March 4, 2010, the most recent version of XF Rendering Server enables the adding of JavaScript code, which is essentional for building rich and especially interactive 3D visualizations, contained in PDF documents. Since PDF 1.6, many more 3D-related features, such as the definition of predefined views and the appending of additional resources have been available, but obviously not yet supported by the XF Rendering Server application from Ecrion. In contrast do their approach defined in a proprietary XML namespace, the work presented in this thesis will support additional 3D-related features, as well as propose a more general method for the integration of 3D content in PDF documents.

3.2 PDF3D - 3D Visualization and Technical Publishing

PDF3D is a commercial suite of software products from Visual Technology Services Ltd. and is designed for the generation of PDF documents, containing 3D objects [49]. This includes collaborative office products for home and business, automated servers and a C++ Software Development Kit (SDK). Based on the PDF3D SDK, existing workflows or applications can be enhanced with a 3D PDF export functionality. This commercial software tool is also able to generate the three-dimensional content itself.

3.3 The movie15 LaTeX Package

In contrast to the overwhelming majority of commercial products, one of the first freely available, open-source tools for generating 3D contained PDF documents is the *movie15* LaTeX package from Alexander Grahn [25]. In addition to movies and sounds, this LaTeX package also provides an interface for embedding 3D content in PDF documents and is able to work with LaTeX as well as pdfLaTeX. Compared to other available tools, poorly exploiting the 3D-related capabilities of

PDF, this package implements a wide range of functions and therefore enables various visualization scenarios. The movie15 package provides support for the following 3D-related features:

- A list of predefined views, including the position and orientation of the virtual camera, as well as the background color, lighting mode and the render mode can be defined.
- The default view, which also supports the previously mentioned view settings, can be specified.
- 3D-related JavaScript code can be added to the 3D stream and will be executed when the resulting 3D artwork becomes active.
- Custom resources can also be attached to the 3D stream and may be loaded dynamically by JavaScript code.
- The rendering mode, the visibility and the transparency of every single part in the scene can be predefined.

Apart from the embedded 3D content itself, the movie15 package also supports the `\movieref` command, which allows the definition of hyperlinks for changing views or executing custom JavaScript code. In contrast to the approach presented in this work, the movie15 package is difficult to use for non-technicians and therefore presupposes a higher level of knowledge. Furthermore, it is quite demanding to integrate 3D content and related meta-information from arbitrary sources.

3.4 Apache FOP

Apache FOP [45], from the Apache Software Foundation, is similar to the XF Rendering Server from Ecrion, an XSL-FO formatter, but does not yet provide the ability to embed 3D objects in an FO document. Since one of the goals of this work is to provide a freely available alternative to commercial PDF generation tools, like for example Adobe Acrobat, and Apache FOP is an open-source project licensed under the Apache License version 2.0, the practical implementation will be based on this XSL-FO formatter. It is implemented in the Java programming language and therefore by default available for multiple platforms, including desktop and (web)server environments. The most recent stable version is 0.95, which implements a large subset of the XSL-FO 1.1 W3C Recommendation and provides support for numerous different output formats, also referred to as *render targets*. The following listing gives a brief overview of supported output formats, available in version 0.95:

- PDF 1.4 (Portable Document Format, published by Adobe)

- PostScript (also published by Adobe)
- PCL (Printer Command Language, contrived by Hewlett-Packard)
- AFP (Advanced Function Presentation, developed by IBM)
- XML (Internal Apache FOP area tree representation encoded in XML)
- Java2D/AWT (Java2D-based output formats like AWT viewer, direct print, PNG and TIFF)
- Other partially supported output formats like RTF (Rich Text Format), plain text, SVG, MIF (*Maker Interchange Format*)

In addition to the standard features of XSL-FO 1.1, almost every available XML formatter defines an own XML namespace for proprietary extensions. The XML namespace for extensions used in Apache FOP is:

```
xmlns:fox="http://xmlgraphics.apache.org/fop/extensions"
```

For example, the XSL-FO 1.1 specification does not, by default, support any kind of rotation for the `fo:block-container` element, but since rotations are often quite useful, the `fox:transform` attribute fills this gap. XSL-FO documents that make use of elements or attributes from this extension namespace are still valid XSL-FO documents, but these proprietary elements and attributes will probably be ignored by other FO processors.

If the default functionality of Apache FOP, including its extension namespace `fox`, is still not sufficient and needs to be extended for some reason, Apache FOP provides an interface for custom-made software extensions too. Referring to its documentation, Apache FOP can be enhanced with the following three types of custom extensions:

- An output document extension such as the PDF bookmarks (has recently been superseded by the bookmark feature of XSL 1.1).
- An `fo:instream-foreign-object` extension such as for SVG, which is supported by default.
- An FO extension that creates an area in the internal area tree of Apache FOP, where normal XSL-FO is not possible.

As shown in Figure 2.2b, the XSL-FO processing workflow is a two-stage process. Since Apache FOP is primarily concentrated on the second stage, the rendering of FO documents, the initial XML transformation is done by the integrated Xalan-Java XSLT processor, which implements XSLT 1.0 and XPath 1.0 [46]. Apache FOP can also be configured to skip the first stage and directly process FO documents, possibly created by another computer program or XSLT processor, such as SAXON that supports XSLT and XPath version 2.0 [37].

Chapter 4

FO3D: 3D Support in XSL-FO

Various approaches for publishing three-dimensional content as well as a small selection of representative metadata element sets and their fields of application have been presented in previous chapters. The main goal of this work is the proposal of a simple yet effective method for integrating both, 3D content and related metadata from arbitrary sources within a single PDF document. At this point it has to be mentioned that the creation of 3D content is not part of this work. Related 3D computer graphics have to be created previously by other tools and should be encoded in one of the supported formats as described in Section 2.1.4. This chapter describes how XML technology, especially XSL-FO, can be used and extended to support the generation of PDF documents that also contain rich 3D content.

4.1 Covering Multiple Sources of Metadata

Referring to the ambitious goals denoted before, the clarification of the term "arbitrary sources of metadata" is the first challenge that has to be considered. The main difficulty is that each and every metadata element set defines properties of different names and meanings depending on its primary field of application.

A very naive approach for covering this versatility would be the creation of a new element set, at least comprising a wide range of properties from other metadata frameworks and providing tools for converting between them. Since there are far too many different available metadata concepts and all available element sets are more or less useful in certain scenarios, this approach seems to be a bottomless pit or will result in a significant loss of flexibility, tightly limiting the field of application.

Obviously, it is not manageable to cover a wide range of metadata frameworks in one generalized element set. Fortunately all metadata element sets described in Section 2.3 and beyond, share at least the following very important quality.

All metadata element sets are per default encoded in, or can be transformed to XML.

Each metadata element set might be serialized to XML in a different manner and some already define their XML representation, respectively their XML vocabulary in their corresponding specification. This work neither assumes any property names, nor the semantic of their values and therefore it is well-suited to support a great variety of different sources.

Among other approaches, a very intuitive and graceful technique for transforming XML-encoded data to PDF documents is XSL-FO, which has been designed to describe the visual presentation of XML documents. Hence, any XML-based metadata element set can be straightforwardly transformed into a mixed content XML document that contains both, the meta information and FO elements describing their visual appearance. An overview of the proposed workflow, from arbitrary XML-based input data to a PDF document on the output side is shown in Figure 4.1. More information on XSL-FO is available in Section 2.2.2.

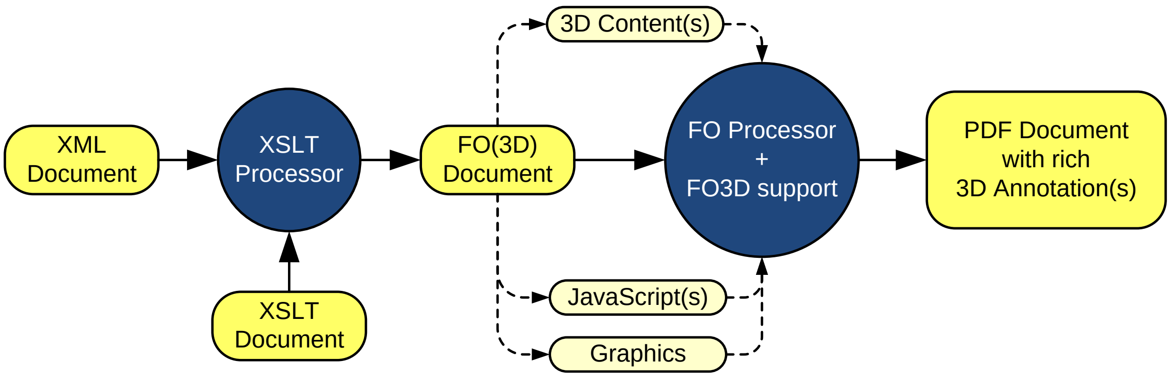


Figure 4.1: Workflow diagram of the proposed XSL-FO formatting process for embedding 3D content in PDF documents.

Since the FO vocabulary is designed for the rendering of two-dimensional layouts only, it is not by default capable of embedding 3D content. At first sight, this might be an exclusion criterion for integrating XML-based metadata with 3D content by means of XSL-FO, but a special FO element provides the ability to handle "generic" XML objects. This specific element from the FO vocabulary is called `fo:instream-foreign-object` and is described more precisely in the following section.

4.2 The XSL-FO "instream-foreign-object"

A brief introduction to the basic concepts of working with XSL-FO has been given in Section 2.2.2, but a specific element needs to be described in more detail. The element of interest is

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
3    <fo:layout-master-set>
4      <fo:simple-page-master page-width="21cm" page-height="29.7cm" master-name="A4">
5        <fo:region-body margin="2cm"/>
6      </fo:simple-page-master>
7    </fo:layout-master-set>
8    <fo:page-sequence master-reference="A4">
9      <fo:flow flow-name="xsl-region-body">
10       <fo:block>
11
12         <fo:instream-foreign-object width="5cm" content-width="scale-to-fit">
13           <svg:svg width="22pt" height="22pt" xmlns:svg="http://www.w3.org/2000/svg">
14             <svg:g style="fill:red; stroke:#000">
15               <svg:rect x="1" y="1" width="15" height="15"/>
16               <svg:rect x="6" y="6" width="15" height="15"/>
17             </svg:g>
18           </svg:svg>
19         </fo:instream-foreign-object>
20
21       </fo:block>
22     </fo:flow>
23   </fo:page-sequence>
24 </fo:root>
```

Listing 4.1: FO document example for directly embedding a SVG document as a descendant of an `fo:instream-foreign-object` element.

named `instream-foreign-object` and according to its specification in the W3C XSL-FO 1.1 Recommendation [13], its common purpose is defined as follows:

> "The fo:instream-foreign-object flow object is used for an inline graphic or other 'generic' object where the object data resides as descendants of the fo:instream-foreign-object, typically as an XML element subtree in a non-XSL namespace."

The most popular example for the use of an `fo:instream-foreign-object` element is the embedding of SVG [33], which is a file format for describing two-dimensional vector graphics on the basis of XML. Listing 4.1 exemplarily shows such an embedded SVG graphics that can directly be rendered by FO processors supporting SVG. This SVG document, from line 13 to 18, describes two interleaved red squares, each surrounded by a black border.

As denoted above, the `fo:instream-foreign-object` element is not only used for embedding graphical, but also for 'generic' formats as long as the FO processor is able to handle the given XML subtree. This means that any valid XML tree from a non-XSL namespace can be embedded in an FO document. The permitted XML structure of the child element is just depending on the vocabulary defined for the applied namespace, for example, SVG or even MathML [50]. The content size of the `fo:instream-foreign-object` is defined by taking the intrinsic size of its child node and the scaling properties, specified by its `content-height`, `content-width` and `scaling` element.

FO processors like Apache FOP [45], XEP from RenderX Incorporated[1] and the Antenna House Formatter[2] are processing descendants of the `fo:instream-foreign-object` by delegating the complete subtree to a module that is familiar with the given XML vocabulary and knows how to visualize the contained data according to the chosen output format.

4.3 Extending XSL-FO with Support for 3D Content

As described in Section 4.2, the `fo:instream-foreign-object` allows the residence of XML subtrees, typically defined in a non-XSL namespace within the (XSL-)FO document itself. This feature opens the door for the integration of 3D content, while preserving standard conformance. Therefore, a custom XML vocabulary will be proposed and explained in the following. A schematic overview of the proposed XML structure is presented in Figure 4.2.

The proposed XML vocabulary, describing 3D related data, is hereafter referred to as *FO3D*, which is the abbreviation of *3D Formatting Object*. Furthermore, a conforming FO processor, supporting FO3D as a subtree of `fo:instream-foreign-object` elements is hereafter referred to as *FO3D processor*.

Since it is planned to release the practical implementation described in Chapter 5 as an open-source project also called *FO3D*[3], the required XML namespace is accordingly defined as follows:

```
xmlns:fo3d="http://fo3d.sourceforge.net/ns"
```

This work is primarily targeted to the integration of 3D content and related metadata in PDF documents, therefore the proposed FO3D vocabulary contains various settings and concepts following the ISO-standardized PDF 1.7 specification (ISO 32000-1 [4]).

In addition to the textual introduction of the FO3D elements, a table that contains an overview of available attributes is presented for every single element. This includes the name, type, short description and a potentially existing default value. The used attribute types are explained briefly in Table 4.1.

4.3.1 3D Object Representation

The topmost element of an FO3D document is named `object-3d`. Possibly the name “3d-object” would be more intuitive, but since the XML specification has clear guidelines concerning the names of elements, respectively attributes, and a digit may not occur in the first place, this is

[1] `http://www.renderx.com`
[2] `http://www.antennahouse.com`
[3] `http://fo3d.sourceforge.net`

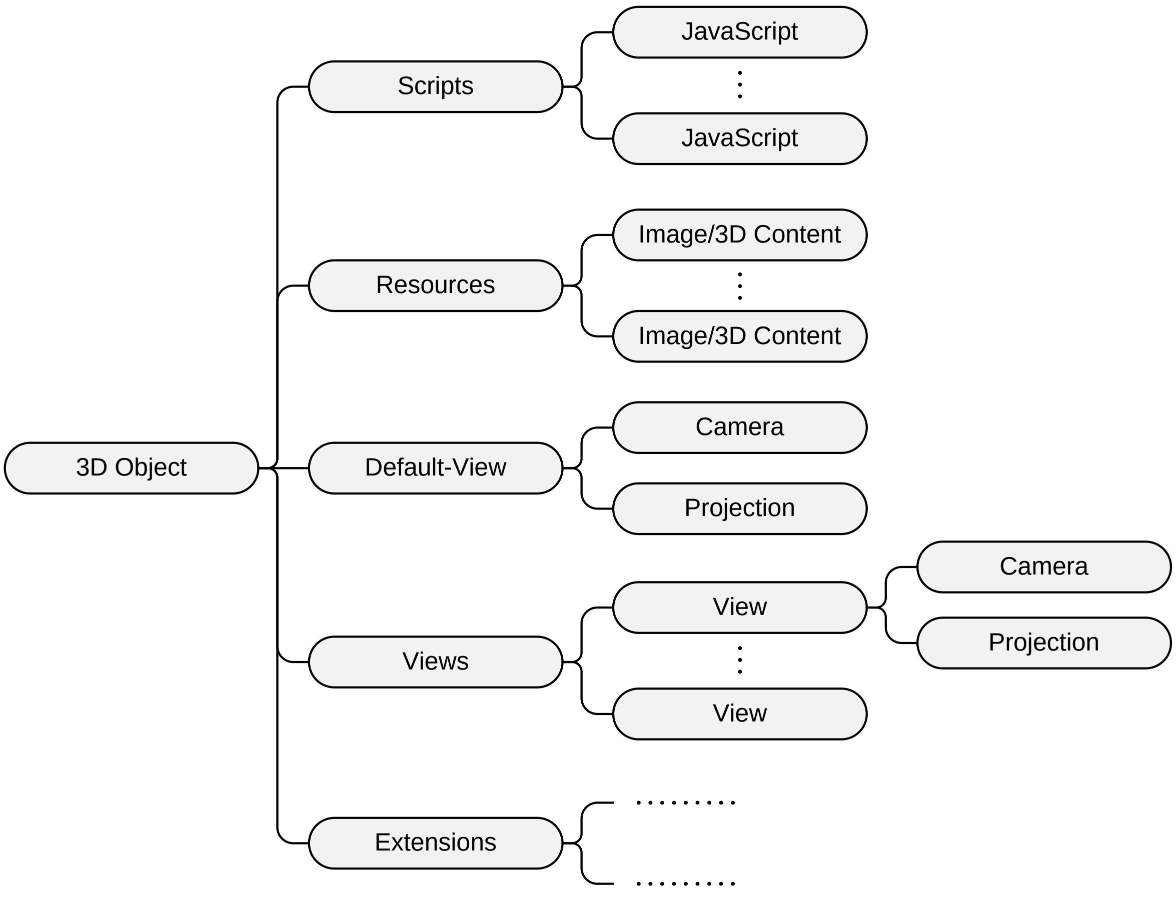


Figure 4.2: Structural overview of the FO3D XML vocabulary.

Type	Description
Boolean	One of the two logical states "true" or "false".
Number	A decimal or integer number that can either be positive, negative or zero. For example: `42`, `0.383`, `0`, `-.42`
Length	A positive decimal or integer number that is followed by one of the following units: `cm`, `mm`, `in`, `pt`, `pc`, `px`
String	Sequence of literal characters without linebreaks.
URI	A Uniform Resource Identifier, primarily used to reference external files.
Enum	Selection of a single value from an enumeration of multiple predefined values.
Color	A color value, similar to those used in the XSL-FO specification. For example: `#FF0`, `#A890E3`, `rgb(255,0,255)`, `rgb(40%,18%,60%)`
TMatrix	This type represents a 3D Transformation Matrix, expressed by a sequence of twelve decimal numbers, which shall be separated by a blank character, a comma or a semicolon. Equation 4.1, on page 43, specifies the order and therefore the meaning of the twelve decimal values.
Vector3	A sequence of three decimal or integer numbers, separated by a blank character, a comma or a semicolon. This type represents a three-dimensional vector used for describing positions and directions in 3D space.

Table 4.1: Overview of FO3D attribute types and their meaning.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:fo3d="http://fo3d.
      sourceforge.net/ns">
3   ...
4     <fo:instream-foreign-object>
5       <fo3d:object-3d src="./model.u3d" width="10cm" height="10cm"/>
6     </fo:instream-foreign-object>
7   ...
8 </fo:root>
```

Listing 4.2: Extending XSL-FO with a simple FO3D document, referencing a U3D file which contains the real 3D content.

not applicable. Line 5 of Listing 4.2 demonstrates how FO3D can easily be integrated in an (XSL-)FO document, as a subtree beneath the `fo:instream-foreign-object`. At once this listing represents the smallest possible FO3D (sub)tree, because the `src` attribute, referencing the real 3D content, and the dimensions, represented by the `width` and the `height` attribute, are mandatory. The entire list of attributes, supported by the `object-3d` tag, is shown in Table 4.2.

By providing the required attributes: `src`, `width` and `height`, an FO3D processor already is in the position to generate a PDF document that contains the corresponding 3D content. Such 3D artworks are, in PDF documents, represented by a *3D annotation dictionary*, referring to a *3D stream dictionary* that contains the real 3D content.

Referring to the PDF specification, multiple 3D annotation dictionaries can refer to the same 3D stream object, which may reduce the resulting file size of the PDF document tremendously. Therefore, FO3D proposes the aggregation of 3D content, based on the URI that is defined by the `src` attribute. This approach allows multiple 3D annotations to refer to a single 3D stream object, for example, providing different default views of the same 3D content. Since 3D-related JavaScript code, which is more precisely described in Section 4.3.5, is directly attached to a 3D stream object, this optimization might result in an unwanted behaviour. This is because it is not possible to create multiple instances of 3D annotations that refer to a single 3D stream and execute different JavaScript code. Therefore, it has to be ensured that the 3D content is added as often as different JavaScript code shall be executed in the corresponding 3D artwork. In FO3D, this is done by setting the boolean attribute `force-embedding` to *true*, which ensures that the related 3D content will definitly be added to the resulting PDF document.

Since initialized 3D annotations require more computational effort, they are disabled by default, but can be activated under certain circumstances. The PDF specification distinguishes between the following three different types of activation and deactivation events:

1. Explict activation/deactivation by the user.
2. The PDF page to which the corresponding 3D annotation is attached to, is loaded/unloaded in the PDF viewer application.
3. The area at which the 3D annotation itself is located becomes visible/invisible in the PDF viewer application.

These three types of activation and deactivation events can also be defined in FO3D documents, by making use of the equally named attributes `activation` and `deactivation`. After its activation/deactivation, the 3D artwork can be in several predefined states, which can be defined by the `after-activation` and `after-deactivation` attributes. For further information on activation/deactivation and the corresponding states afterwards, it is referred to the specification of the PDF *3D Activation Dictionary*, which is directly referred to by the *3DA* key of the 3D annotation dictionary.

In case the 3D artwork is deactivated, an alternative appearance can be defined for the rectangular area, the 3D content is normally being displayed. With the FO3D `altsrc` attribute, a two-dimensional graphics can be defined for this purpose. In PDF documents, this behavior is realized by a so-called *Appearance Dictionary*, which refers to a separate content stream containing the 2D graphics.

Apart from just defining activation/deactivation events and corresponding states afterwards, the 3D activation dictionary further provides several keys for enabling/disabling the interactive toolbar (`toolbar` attribute) and the default visibility of an interface for viewing or managing information about the 3D content (`modeltree` attribute). Since PDF 1.7 Adobe Extension Level 3 (Adobe Acrobat 9), this dictionary also enables the definition of a transparent background. In FO3D, this can be defined by using the `transparent` attribute of the `object-3d` element. To disable all interactive features including the previously mentioned toolbar, the attribute `interactive` can be set to *false*, which deactivates all user interaction features with the 3D content.

Since the beginning of 3D support in PDF 1.6, 3D annotations cannot only be linked directly to a 3D stream, but also by using a so-called *3D Reference Dictionary.* This dictionary can be referenced by multiple 3D annotations and itself refers to a single 3D stream object. Multiple 3D annotations that are referring to the same 3D reference dictionary share the same run-time instance and therefore, interaction in one 3D annotation is reflected in the viewport of the other 3D annotations.

Name	**Type**	**Description**	**Default**
src	URI	*(Required)* Refers to an external 3D content file (U3D or PRC).	
width	Length	*(Required)* Width of the rectangular area in which the 3D artwork is displayed.	
height	Length	*(Required)* Height of the rectangular area in which the 3D artwork is displayed.	
name	String	*(Optional)* Internal name of the PDF 3D annotation, used for the unique identification, for example, by JavaScript code.	
altsrc	URI	*(Optional)* Reference to a 2D graphics, that is used as an alternative overlay, if the PDF 3D annotation is deactivated, and can also be used for other strictly two-dimensional render targets.	
refgroup	String	*(Optional)* This attribute indicates that the corresponding 3D annotation and the 3D stream shall be linked over a PDF *3D Reference Dictionary*. Together with the `src` attribute, the *3D Reference Dictionary* can uniquely be identified and referred to from multiple 3D annotations, which then share the same run-time instance of the corresponding 3D stream.	
activation	Enum	*(Optional)* A name specifying the circumstances under which the 3D annotation shall be activated. Options: *user, page, visibility*	user
after-activation	Enum	*(Optional)* A name specifying the state of the artwork instance upon activation of the 3D annotation. Options: *instantiated, live*	live
deactivation	Enum	*(Optional)* A name specifying the circumstances under which the 3D annotation shall be deactivated. Options: *user, page, visibility*	visibility
after-deactivation	Enum	*(Optional)* A name specifying the state of the artwork instance upon deactivation of the 3D annotation. Options: *uninstantiated, instantiated, live*	uninstantiated
force-embedding	Boolean	*(Optional)* Whether a 3D stream object shall be explicitly created for the current 3D annotation (larger files) or not. If set to *false*, FO3D objects that share the same `src` URI will be represented by a single 3D stream object, regardless of how often it is referenced by 3D annoation objects.	false
interactive	Boolean	*(Optional)* Whether the 3D scene can be manipulated interactively or not. If set to *false*, the 3D annotation can only be manipulated programmatically.	true
toolbar	Boolean	*(Optional)* Whether an interactive toolbar associated with the 3D artwork shall be displayed or not. If `interactive` is set to *false*, no toolbar will be shown either.	true
modeltree	Boolean	*(Optional)* Whether an interface for viewing or managing information about the 3D artwork shall be displayed or not.	false
transparent	Boolean	*(Optional)* Whether the background of the 3D annotation shall be transparent or not. This feature is available since PDF 1.7 Adobe Extension Level 3.	false

Table 4.2: Attribute overview of the topmost `fo3d:object-3d` element.

FO3D support for this feature is provided by the `refgroup` attribute, which indicates that the 3D annotation and the corresponding 3D stream object shall be linked over such a 3D reference dictionary. The combination of `src` and `refgroup` attribute uniquely identifies such a 3D reference dictionary, so that multiple 3D annotations can share the same run-time instance of the related 3D stream object. Since the corresponding 3D content stream is just added once to the resulting PDF document, additional resources have to be specified with the first occurence of such an `object-3d` element. For example, the list of predefined views, 3D related JavaScript code and additional resource files, all described in the following sections shall be ignored for subsequent `object-3d` elements in the FO document. Since the FO3D `default-view` element described in Section 4.3.2 is responsible for the definition of the default view of a 3D annotation dictionary and is not directly linked to a 3D stream object, it can be defined separately for all such 3D annotations, grouped by the `refgroup` attribute and share the same run-time instance. Section 6.4.4 provides additional information, especially for Adobe Acrobat, and an observed pitfall in dealing with such 3D reference dictionaries.

Finally, the `name` attribute provides the ability to define an internal name for the resulting 3D annotation. This name shall be unique and can, for example, be used to access the 3D annotation object programmatically, by means of JavaScript code.

4.3.2 Predefined 3D Views

As mentioned previously, 3D annotations can contain multiple views in order to present some particular parts of the 3D content more precisely or even manipulate some settings that are relevant to the visual appearance of the 3D-content. Therefore, FO3D provides the `default-view` and the `views` element, which are both direct descendants of the `object-3d` node. The `views` element has no attributes and just serves as a container for multiple elements, named `view`. Such a `view` element is very similar to the `default-view` element and finally both are used for the creation of PDF *3D View Dictionary* objects. The `view` element supports all attributes that are availabe for the `default-view` element and additionally defines some more. Both elements can also contain the `camera` and `projection` element as direct descendants. The `camera` element, explained in more detail in Section 4.3.3, is responsible for the position and the orientation of the virtual camera in 3D space. With the `projection` element described in Section 4.3.4, several parameters of the two-dimensional projection to the viewport can be influenced. The `default-view` element, as well as the list of `view` nodes and their relevance to 3D artworks are described in the following.

An example that shows all of the previously mentioned elements is presented in Listing 4.3. Line 4 shows the `default-view` element, which defines a dark gray background and sets the visual appearance of 3D objects to *transparent*. The `views` element, starting in line 8, contains two predefined views, where the first one changes the lighting mode and can be accessed by its name, and the second one influences the virtual camera, as well as the 2D projection mode.

```
<fo:instream-foreign-object>
  <fo3d:object-3d src="model.u3d" width="10cm" height="10cm">

    <fo3d:default-view background-color="#333" render-mode="transparent">
      <fo3d:camera position="12.38, 22, .28" target="29.9, 12.1, 4" up="0,1,0"/>
    </fo3d:default-view>

    <fo3d:views>
      <fo3d:view title="View 1" name="v1" light-mode="cad"/>
      <fo3d:view title="View 2">
        <fo3d:camera c2w="1 0 0 0 1 0 0 0 1 0 0 0"/>
        <fo3d:projection type="perspective" fov="30"/>
      </fo3d:view>
    </fo3d:views>

  </fo3d:object-3d>
</fo:instream-foreign-object>
```

Listing 4.3: Exemplary use of `default-view`, `view` and `camera` elements.

The Default View

The FO3D `default-view` element is used for the explict definition of a default view for the PDF 3D annotation dictionary and is referred to by its *3DV* key. Therefore, the default appearance of the 3D annotation can be defined independently from the default view that may be defined for the related 3D stream object. Concerning the `refgroup` attribute of `object-3d`, the `default-view` may be expedient for setting up multiple 3D annotation objects that share a single 3D stream object and initially show the same scene in totally different views.

Furthermore, the `default-view` supports various attributes that are presented in Table 4.3. Starting with the `background-color` attribute, the whole viewport can be colored in the back. Similar to the `color` and `background-color` attributes known from the XSL-FO vocabulary, the FO3D `background-color` attribute of the `default-view` and `view` element respectively are supporting various color definitions too. For example, the hexadecimal forms `#F0F` and `#3A83D0` or RGB statements in the form of `rgb(255,90,10)` and `rgb(20%,50%,4%)` are supported.

In contrast to the `background-color` attribute which does not affect the 3D content itself, the `light-mode` and the `render-mode` attributes allow a more detailed definition of how the 3D content shall be displayed. Both, the value for `light-mode` and `render-mode` have to be selected from a predefined list of available options, denoted in Table 4.3. For more detailed information on these options, it is referred to the PDF specification of the *3D Lighting Scheme Dictionary* and *3D Render Mode Dictionary*, respectively [4].

Concerning the PDF 3D render mode dictionary, the PDF reference allows a more detailed specification for various settings, such as the opacity for *transparent* renderings or the crease angle that shall be used for determining silhouette edges in *illustration* render modes. Currently, these settings are not available in form of FO3D view attributes, but can be set by appropriate JavaScript code. How JavaScript code, by means of FO3D, can be added to 3D artworks is more

Name	Type	Description	Default
background-color	Color	*(Optional)* Background color of the 3D artwork. (if not set to transprent)	
render-mode	Enum	*(Optional)* Render mode of the current view. More information on the available options is provided in the corresponding PDF reference. Options: *solid, transparent, wireframe, boundingbox, vertices, illustration, solid-wireframe, transparent-wireframe, shaded-wireframe, hidden-wireframe, transparent-boundingbox, transparent-boundingbox-outline, shaded-vertices, solid-outline, shaded-illustration*	
light-mode	Enum	*(Optional)* Lighting mode of the current view. For more information on the available lighting options, it is referred to the PDF specification. Options: *none, artwork, white, day, night, hard, primary, blue, red, cube, cad, headlamp*	artwork

Table 4.3: Attribute overview of the `fo3d:default-view` element.

precisely explained in Section 4.3.5.

List of Predefined Views

FO3D `view` elements, which are all descendants of the `views` node, are in the resulting PDF document attached to the 3D stream object itself, by using its dictionary key *VA* that represents an array of indirect references to corresponding *3D View Dictionary* objects. Some attributes of the `view` have already been introduced to the `default-view` element, shown in Table 4.3. In addition to them, the `view` element also provides support for the `title`, `name` and `default` attributes, which are listed in Table 4.4.

Referring to the PDF specification, the list of 3D view dictionaries that is attached to the 3D stream object is shown in a user interface of a conforming PDF viewing application. Therefore, the `view` element requires an external name, which has to be specified by the mandatory `title` attribute. In addition to the external name, 3D view objects can optionally define an internal name too, which is represented by the `name` attribute. This internal name can, for example, be used for a PDF *Go-To-3D-View Action* or for accessing the corresponding view object by means of JavaScript.

Similar to the PDF 3D annotation, a default view can also be specified for a 3D stream dictionary. In FO3D, this can be done by setting the attribute `default` of a `view` element to *true*. Obviously only one element in the whole list of `view` nodes should be marked as default view.

The `default` attribute of the `view` elements is ignored, if the `object-3d` element does explicitly define a `default-view`. If no `default-view` and also no default `view` element is specified, the first `view` from the list will be used.

Name	Type	Description	Default
title	String	*(Required)* External name of the view, presented in the user interface of a conforming PDF viewing application, such as the interactive toolbar of Adobe Reader.	
name	String	*(Optional)* Internal name, used for accessing this view object by Go-To-3D-View actions or JavaScript code.	
default	Boolean	*(Optional)* Whether this view shall be used as the default view for the corresponding 3D stream object or not. Since only one view can be used by default, the list of predefined views shall at most contain one `view` element that has a `default` attribute set to *true*.	false

Table 4.4: Additional attributes of the `fo3d:view` element.

4.3.3 Camera

Along with 3D comes the concept of a virtual camera, which is in theory just a point in 3D space that provides a two-dimensional view of the 3D scene. Since a single point is just defining the position and not where the camera is looking at, its orientation has to be defined too. Usually, this is done by providing a 4×4 square matrix, which exactly defines the transformation between 3D world space coordinates and the coordinate system of the virtual camera.

As mentioned in Section 4.3.2, every FO3D `default-view` and `view` element can optionally contain a `camera` element as a descendant, defining what particular part of 3D space is presented in the viewport. The entire list of attributes supported by the `camera` element is presented in Table 4.5.

Although the transformation matrix of the virtual camera may, referring to the PDF specification, also be directly defined in the 3D content itself, the FO3D `camera` element explicitly specifies the position and the orientation of the virtual camera. As shown in line 11 of Listing 4.3, the `c2w` attribute defines the Camera-To-World (C2W) transformation matrix by a sequence of twelve numbers. Equal to the PDF spceification, these numbers shall be specified in a predefined order, which is defined in equation 4.1. The C2W matrix is equal to the inverse of the more commonly known *View Matrix* and can, in PDF documents, be specified by the *C2W* key of the corresponding 3D view dictionary.

$$\begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d & e & f & g & h & i & t_x & t_y & t_z \end{bmatrix} \tag{4.1}$$

It has to be mentioned that 3D artworks in PDF documents are by default based on a left-handed 3D Cartesian coordinate system, which means that the orientation of the three axes can be simulated by thumb (x), forefinger (y) and middle finger (z) of the left hand. By default the camera is located at (0, 0, 0) and the x-values increase to the right, the y-values increase

Name	Type	Description	Default
c2w	TMatrix	*(Optional)* 4×4 Camera-To-World transformation matrix, represented by a sequence of 12 decimal numbers, according to the order that is explained in equation 4.1.	
position	Vector3	*(Optional)* A point in 3D space (world coordinates), the camera is located at. (Ignored if the `c2w` attribute is used)	
direction	Vector3	*(Optional)* 3D vector defining the viewing direction of the camera in world coordinates. (Ignored if the `c2w` attribute is used)	
up	Vector3	*(Optional)* 3D Vector specifying the upper edge of the viewport, in world coordinates. (Ignored if the `c2w` attribute is used)	
target	Vector3	*(Optional)* 3D Vector defining a point in 3D space (world coordinates), the camera is looking at. (Ignored if the `c2w` attribute is used)	
co	Number	*(Optional)* Non-negative number representing a distance along the z-axis in the camera coordinate system, which defines the *Center of Orbit*, used for rotation interaction tools.	

Table 4.5: Attribute overview of the `fo3d:camera` element.

upward and positive z values increase going into the page. In contrast to PDF, the U3D file format is referring to its specification [20] wherever applicable and not declared otherwise, based on the right hand rule for orienting the three axes, but conformable PDF applications handle this difference automatically.

XML is often praised to be a human readable format, but it is quite cumbersome to extract or set the position and orientation of the virtual camera by a sequence of twelve decimal numbers. Therefore FO3D also provides an alternative method. A position, direction and up vector specifying the upper edge of the resulting viewport can also be defined instead of the C2W matrix. For this purpose, the `camera` element provides the analogously named 3D vector attributes `position`, `direction` and `up`. As an alternative to the `direction`, the `target` attribute specifies a point in 3D space, where the camera is looking at. The direction vector can straightforwardly be computed again by subtracting the defined position from the given target vector.

Given the position $\mathbf{p}$, the direction vector $\mathbf{d}$ and the up vector $\mathbf{u}$, the C2W matrix and therefore the twelve-element array of numbers can be calculated as shown in equation 4.2 to 4.4.

Apart from the described attributes that are used to define the position and the orientation of the virtual camera, the `co` attribute defines an important property for interactive navigation in 3D space. The abbreviation “co” stands for *Center of Orbit* and this attribute reflects the equally named *CO* key of the PDF 3D view dictionary. The value of the `co` attribute shall be a non-negative number, representing a distance along the z-axis in the camera coordinate system. The resulting point in 3D space is used as the point around which the camera rotates when performing an orbit-style navigation.

$$\mathbf{d} = \frac{\mathbf{d}}{\|\mathbf{d}\|}, \qquad \mathbf{u} = \frac{\mathbf{u}}{\|\mathbf{u}\|} \tag{4.2}$$

$$\mathbf{s}' = \mathbf{u} \times \mathbf{d}, \qquad \mathbf{s} = \frac{\mathbf{s}'}{\|\mathbf{s}'\|}, \qquad \mathbf{v} = \mathbf{d} \times \mathbf{s}, \tag{4.3}$$

$$C2W = \begin{bmatrix} \mathbf{s}_x & \mathbf{s}_y & \mathbf{s}_z & 0 \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & 0 \\ \mathbf{d}_x & \mathbf{d}_y & \mathbf{d}_z & 0 \\ \mathbf{p}_x & \mathbf{p}_y & \mathbf{p}_z & 1 \end{bmatrix} \tag{4.4}$$

4.3.4 2D-Projection

In PDF documents, the two-dimensional projection settings are represented by a *Projection Dictionary*, which is referred to by the *P* key of a 3D view dictionary. FO3D therefore introduces the `projection` element, which can be used as a direct descendant of the `default-view` or the `view` element. The entire list of available attributes is presented in Table 4.6.

The most important attribute, supported by the `projection` element, is called `type`. It is mandatory, because other attributes of the same element are depending on its value. Since the PDF specification defines two different kinds of two-dimensional projections, the `type` attribute can either be set to *perspective* or *orthographic*.

The Perspective Projection requires, at least in PDF documents, the *Field Of View* (FOV) property, which defines a cone that is centered around the viewing direction of the virtual camera (z-axis in camera coordinates). Therefore, the `fov` attribute is mandatory when using this projection type.

The Orthographic Projection belongs to the class of parallel projections. The lines of projection are parallel in both reality and in the projection plane itself, which made this the type of choice for working drawings.

Similar to other 3D graphics frameworks, a method for reducing the computational effort in large scenes, called *clipping*, is also available for 3D artworks that are embedded in PDF documents. Hence, a *near* and a *far* clipping plane is introduced and only those objects that are located between these two planes are finally displayed in the viewport. For this purpose, the `projection` element provides three attributes, called `clipping`, `near` and `far`. Referring to the PDF reference, the `clipping` attribute can be used to distinguish between *automatic* and *explicit* clipping. By using the explict clipping, the `near` and the `far` attribute define the position

Name	Type	Description	Default
type	Enum	*(Required)* 2D projection type of the corresponding view. Options: *perspective, orthographic*	
fov	Number	*(Required for perspective, ignored for orthographic projections)* This attribute represents the Field Of View of the virtual camera and its value can be specified between 0 and 180 degrees.	90
clipping	Enum	*(Optional)* The type of clipping, that shall be applied for the current view. Options: *auto, explicit*	auto
near	Number	*(Required for perspective projections if explict clipping is used)* Positive number, defining the position of the near clipping plane, if explicit clipping is used. Otherwise this attribute is ignored.	0
far	Number	*(Optional)* Positive number that defines the distance between virtual camera and far clipping plane, if explicit clipping is used. Otherwise this attribute will be ignored. If this attribute is absent, no far clipping will occur.	

Table 4.6: Attribute overview of the `fo3d:projection` element.

of the equally named near and far clipping plane in 3D space, orthogonally to the z-axis in the camera coordinate system. In case of automatic clipping which is used by default, the `near` and `far` attribute is ignored and a conforming PDF viewing application shall, based on the number of 3D objects in the scene, automatically adjust the position of both planes. For a more detailed explanation of clipping and the corresponding near and far clipping plane, it is referred to the specification of the projection dictionary in the PDF reference.

4.3.5 JavaScript Support

Since PDF added support for JavaScript as a client-side scripting language, such documents became more and more interactive. Once primarily intended for the use with forms, the PDF JavaScript APIs today are providing access to almost every object available in a PDF document. Especially 3D annotations can benefit from JavaScript, providing more flexibility and support for interactive visualizations. Since the main goal of this work is the integration of 3D content and related metadata, JavaScript nearly provides boundless freedom in defining scenarios that can benefit from this synergy. Hence, JavaScript is the key to a successful fusion of 3D content and related semantic metadata.

FO3D proposes a simple method for adding JavaScript code to 3D artworks and also to enrich PDF documents with standard document-context JavaScript code. Therefore, three new elements are introduced. The first one, named `scripts`, is a structural element that serves as a container for the two other elements, called `script` and `doc-script`. Both, the `script` and the `doc-script` element can occur multiple times as direct descendants of the `scripts` element. An

```
<fo:instream-foreign-object>
  <object-3d src="model.u3d" width="10cm" height="10cm" xmlns="http://fo3d.
      sourceforge.net/ns">

    <scripts>
      <script src="./js/my-animation.js">
          /* 3D JavaScript API */
          host.app.alert('3D annotation initialized!');
      </script>
      <doc-script type="close">
        /* Standard Acrobat JavaScript API */
        app.alert('PDF says goodbye!');
      </doc-script>
      <doc-script type="open" src="./js/init.js"/>
    </scripts>

  </object-3d>
</fo:instream-foreign-object>
```

Listing 4.4: JavaScript integration example

Name	Type	Description	Default
src	URI	*(Optional)* Reference to an external JavaScript file whose code is added to the 3D JavaScript stream that is executed during the initialization of the corresponding PDF 3D artwork.	

Table 4.7: Attribute overview of the `fo3d:script` element.

expressive example is provided in Listing 4.4. A more detailed description of the `script` and the `doc-script` element is given in the following.

3D JavaScript Support

The FO3D `script` element is used to improve 3D visualizations in PDF documents and supports the integration of JavaScript code, according to the 3D JavaScript API [2]. Similar to the integration of JavaScript code in (X)HTML documents and as shown in Listing 4.4, the FO3D `script` element distinguishes between two different methods for adding JavaScript code to the 3D artwork. The JavaScript code can either be directly inserted as the content of a `script` element, or it can be loaded from an external JavaScript file, using the `src` attribute. Altough the `script` element has just one attribute, the attribute overview is presented in Table 4.7 for the sake of completeness.

Both methods can be combined in a single `script` element, concatenating the external and the internal JavaScript code. Since external JavaScript files more likely contain a collection of reusable object and function definitions, it makes sense to execute it prior to the internal JavaScript code, which possibly is based on the externally defined code. Similar to this concatenation, the JavaScript code of multiple `script` elements is also concatenated in the order it appears beneath the `scripts` container element. Hence, it is also possible to load more than one external JavaScript file to a particular 3D stream.

Name	Type	Description	Default
type	Enum	*(Required)* The type of the occurring event, for which this piece of JavaScript code shall be executed. Options: *open, close, page-open, page-close*	
src	URI	*(Optional)* Reference to an external JavaScript file.	

Table 4.8: Attribute overview of the `fo3d:doc-script` element.

In PDF documents, 3D-related JavaScript code is represented by a PDF stream object, which is referred to by the *OnInstantiate* key of the corresponding 3D stream dictionary. As this dictionary key suggests, a conforming PDF viewer application will execute the JavaScript code, when a corresponding 3D annotation is initialized.

Document-Context JavaScript Support

Similar to the JavaScript code that can be defined for a 3D stream object, the `doc-script` element provides the ability to add JavaScript code that will be executed in the context of a PDF document. The `doc-script` element also allows the inline definition of JavaScript code, as well as the inclusion of an external JavaScript file, referenced by the `src` attribute. In contrast to the `script` element that expects JavaScript code that is based on the 3D JavaScript API, the `doc-script` element accepts document-context JavaScript code, such as defined for Adobe Acrobat in [3].

Since document-level JavaScript code can be used in response to various occurring events, the `type` attribute explicitly specifies when the corresponding code shall be executed. Concerning the integration of 3D objects and PDF documents, not all avialable events are of great interest, but the self-explanatory subset of *open*, *close*, *page-open* and *page-close* events at least covers a wide range of imaginable scenarios. The definition of the `type` is, in contrast to the `src` attribute, mandatory.

The small collection of two attributes, supported by the `doc-script` element, and their corresponding explanation is shown in Table 4.8.

The most interesting event might be the opening of a PDF document (type: *open*), because it is often used for initialization purposes and the definition of additional JavaScript functions that can be used later. This type of JavaScript code is added to the PDF document by directly referring to the JavaScript stream object with the *OpenAction* key of the root PDF catalog dictionary. JavaScript code that shall be executed if the PDF document is closed, is specified in a so-called *Additional-Actions Dictionary*, again referred to by the catalog dictionary. JavaScript code that shall be executed when a PDF page is opened or closed in the viewer application, is also specified in an *Additional-Actions Dictionary*, but in this case it is referenced by the corresponding PDF page object.

Name	Type	Description	Default
src	URI	*(Required)* Reference to an external file, containing 3D content or a 2D computer graphics.	
name	String	*(Required)* Unique identifier, that can be used for accessing this resource in 3D JavaScript.	

Table 4.9: Attribute overview of the `fo3d:resource` element.

4.3.6 Additional Resources

A 3D object can be enriched tremendously by adding additional resources that are not part of the original 3D content itself. The 3D stream dictionary therefore defines the *Resources* key, which can refer to a PDF *Name Tree*, mapping name strings to objects. These objects can be loaded by JavaScript code that is, for example, executed during the initialization of the corresponding 3D annotation. Since not all available PDF objects are suitable for the use in a 3D environment, only two-dimensional graphics and additional 3D objects can be used. The 3D JavaScript API provides several routines, so that an attached image resource can, for example, dynamically be loaded and applied as a texture, or additional 3D content can be placed anywhere in the existing scene.

```
<fo:instream-foreign-object>
  <fo3d:object-3d src="model.u3d" width="10cm" height="10cm">

    <fo3d:resources>
      <fo3d:resource src="http://example.com/ball.u3d" name="ball"/>
      <fo3d:resource src="../img/face.jpg" name="tex"/>
      <!-- ... -->
    </fo3d:resources>

  </fo3d:object-3d>
</fo:instream-foreign-object>
```

Listing 4.5: Adding resources (3D objects or images) to the PDF 3D annotation

The FO3D `resources` element serves as a container for multiple child elements, having the name `resource`. An overview of attributes that can be defined for a `resource` element is presented in Table 4.9 and an example of how resources are defined in FO3D is shown in Listing 4.5.

The `src` attribute is used to reference an external two- or even three-dimensional computer graphics and the `name` attribute provides a unique identifier for the current resource, which is used for the dynamic instantiation by means of JavaScript code. Both, the `src` and the `name` attribute are mandatory.

4.3.7 Custom FO3D Extensions

By using XSLT as a language for transforming arbitrary XML input, FO as a language for describing the visual appearance of a document and FO3D as a tool for providing additional support for 3D content, a wide range of rich and interactive PDF 3D visualizations can be

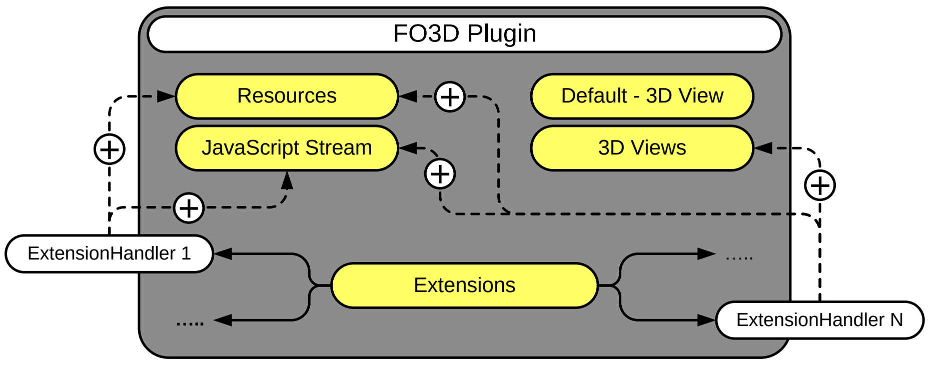


Figure 4.3: Schematic overview of an FO3D plugin, showing its internal components and the cooperation with custom extension handlers.

realized. However, in some cases the proposed workflow might still not be powerful enough. For example, by means of XSLT and JavaScript, it is not possible to generate a binary image or load any arbitrary external file that is referenced from the XML input and shall affect the visual appearance of the 3D content. Therefore, additional support during the PDF generation process has to be provided.

The FO3D `extensions` element provides the ability of storing custom XML elements, without specifying any structural or content-related restrictions. In general, the purpose of this element is similar to the `fo:instream-foreign-object` element, which also provides a method for extending the FO document with custom XML data, such as SVG or even FO3D. In contrast to the `fo:instream-foreign-object`, the descendants of the `extensions` element are defined in the same FO3D namespace and the number of child nodes is not limited to one.

Like an FO processor does not understand the proposed FO3D vocabulary by default, a corresponding FO3D processor plugin is also not by default able to process descendants of the `extensions` element. Therefore, a conforming FO3D processor needs to provide an interface for the dynamic registration of additional handlers. Such a handler, hereafter referred to as *FO3DExtHandler*, is registered by the name of the element it is providing support for. Since a plugin is typically written in same language as its host program, the FO3D plugin and the FO3DExtHandlers might use the same programming language as the underlying XSL-FO processor itself.

A conforming FO3D plugin calls the corresponding FO3DExtHandler for every single element that is a child node of the `extensions` element, having the supported and registered element name. When calling such an FO3DExtHandler, the corresponding element, its attributes and descendants are passed to it. In addition to that, an FO3DExtHandler also has access to the internal representation of all previously decribed PDF primitives, such as the 3D annotation, the 3D stream, the default view, the list of predefined views, the list of resources and the JavaScript streams. A schematic overview of an FO3D plugin, its internal components and the cooperation with arbitrary FO3DExtHandlers is outlined in Figure 4.3.

```
<fo:instream-foreign-object>
  <object-3d src="board.u3d" width="10cm" height="10cm" xmlns="http://fo3d.
      sourceforge.net/ns">

    <extensions>
      <pgn src="http://www.example.com/chessgame.pgn"/>
    </extensions>

  </object-3d>
</fo:instream-foreign-object>
```

Listing 4.6: Exemplary usage of the proposed `fo3d:extensions` element.

In Listing 4.6, the exemplary use of the proposed FO3D `extensions` element is shown. In this example, an external file containing a chess game, which is encoded in the *Portable Game Notation* (PGN) format [24], is referenced. A corresponding FO3DExtHandler could, for example, attach 3D chessmen models as additional resources, then load the external PGN file and transform all contained chess moves to JavaScript animations on a virtual chessboard in 3D space.

Chapter 5

FO3D: An Apache FOP Plugin

This chapter describes a prototype implementation, which extends the wide-spread Apache FOP with a custom `fo:instream-foreign-object` extension to provide additional support for the FO3D vocabulary that has been described in Chapter 4. A short description of how Apache FOP can be enhanced by custom-made extensions is presented in Section 5.1, directly followed by a more detailed explanation regarding the FO3D extension itself. Finally, a few additional enhancements to enable bidirectional linking between the embedded 3D content and the resulting PDF document are presented in Section 5.3.

5.1 Extending Apache FOP

Apache FOP provides various methods for extending its standard functionality and therefore, additional elements, renderers and handler modules for custom XML vocabularies can be added. The following explanation is based on Apache FOP, version 0.95, which is currently the most recent stable version [45]. For building a custom Apache FOP extension, such as needed to add support for the proposed FO3D vocabulary, the Java package `org.apache.fop.fo.extensions` and especially `org.apache.fop.fo.extensions.svg`, are good resources to start from. In the following, three different types of Apache FOP extensions are described more precisely.

Custom XML elements are accepted by Apache FOP, after they and their corresponding Java object representation have been registered. The following enumeration explains how this is done without manipulating the original source code itself.

1. At first, a class that extends the abstract `org.apache.fop.fo.ElementMapping` class has to be created. The `ElementMapping` represents a static hashmap, which is loaded during the bootstrap initialization and contains all elements that are defined in a particular XML

namespace. This makes it easier for Apache FOP to create a custom object for every element.

2. To make Apache FOP aware of the newly created class, the file: "*/META-INF/services/org.apache.fop.fo.ElementMapping*" has to be created and it shall contain the fully qualified classname of the new `ElementMapping` implementation. During its initialization, Apache FOP will look for such files in the classpath and dynamically load the classes that are referred to by their fully qualified classnames.

3. Finally, a *Java Archive* (JAR) has to be created and the resulting JAR file has to be put in the classpath. When starting Apache FOP now, the element Java classes that are referred to from the custom `ElementMapping` class, will be registered.

XML handlers are defined for a specific XML namespace and are able to process corresponding elements and attributes. Apache FOP, by default, contains an XML handler, which is responsible for processing XML data that is defined in the SVG namespace. XML data from a non-XSL namespace, such as SVG, can typically be directly embedded in XSL-FO documents, as descendants of the `fo:instream-foreign-object` element, which has already been introduced in Section 4.2. The creation of a custom XML handler is as simple as the above described creation of an element mapping class. Again a new class has to be created, but in this case, it shall implement the `org.apache.fop.render.XMLHandler` interface. To inform Apache FOP about the existence of this new XML handler class, the fully qualified Java class name has to be stored in the file: "*/META-INF/services/org.apache.fop.render.XMLHandler*".

Renderers can also be customized by implementing the `org.apache.fop.render.Renderer` interface, or just extending the abstract class `org.apache.fop.render.AbstractRenderer`, which already does most of what is needed and the subclass just has to define the representation of primitives like *Page*, *Viewport*, *Region*, *Span*, *Block*, *Line* and *Inline*, according to the dedicated output format. Similar to the definition of a custom element or XML handler, an additional `Renderer` class is dynamically registered by storing its fully qualified Java class name in the file: "*/META-INF/services/org.apache.fop.render.Renderer*".

5.2 Support for FO3D

Based on the previously described extension interfaces of Apache FOP, support for custom XML data from the FO3D namespace is added, so that it is finally possible to create PDF documents, containing 3D artworks.

5.2.1 General Integration

The Java classes that extend Apache FOP for the use with the FO3D vocabulary are generally located in the `net.sourceforge.fo3d` package and its sub-packages. The integration of the FO3D plugin is explained, step by step, in the following.

Transforming an FO document that contains FO3D data, by using the standard Apache FOP, will show a warning that indicates that the FO processor does not know what to do with the given FO3D data and therefore ignores it. For example, the warning for the FO3D namespace and the unknown `object-3d` element looks as follows:

```
23.04.2010 15:21:53 org.apache.fop.fo.ElementMappingRegistry findFOMaker
WARNUNG: Unknown formatting object http://fo3d.sourceforge.net/ns^object-3d
```

To make Apache FOP aware of FO3D elements, the Java class `FO3DElementMapping` registers the following two classes, covering all elements from the FO3D vocabulary that have been introduced in Chapter 4:

- The `FO3DObj` class extends the abstract `XMLObj` class of Apache FOP and is registered as the general representation of any element that is defined in the FO3D namespace. This class just informs Apache FOP about the supported XML namespace and the default namespace prefix. Other necessary methods are already defined in the parent `XMLObj` class.
- The `FO3DElement` extends the general `FO3DObj` class and represents a specialized implementation for the topmost `fo3d:object-3d` element. This implementation is also straightforward, but this class additionally overloads the `getDimension` method, which in Apache FOP is required for child nodes of the `fo:instream-foreign-object` element and used to determine its intrinsic dimensions. In case of FO3D, the dimensions are specified by the `width` and `height` attribute of the `object-3d` element.

Apache FOP now accepts FO3D elements and is able to hold the corresponding Java objects in his internal tree structures. Trying to transform the FO document now will not cause any errors, but the FO3D data will be ignored because Apache FOP does not know what to do with it and therefore just shows the following warning:

```
23.04.2010 16:05:08 org.apache.fop.render.AbstractRenderer renderXML
WARNUNG: Some XML content will be ignored.
         No handler defined for XML: http://fo3d.sourceforge.net/ns
```

Obviously, the next step is the registration of an XML handler that is capable of processing FO3D data. Therefore, the `FO3DPDFHandler`, which implements the `XMLHandler` Java interface, is added to Apache FOP as an extension. The `XMLHandler` interface is straightforward and declares just three methods, which are explained in the following listing:

- The `getNamespace` method informs Apache FOP during the registration procedure of the corresponding XML handler class, which XML namespace it supports.
- Later, during the formatting process, the XML handler will be called for every XML element of the namespace it is registered for. Since Apache FOP allows the registration of multiple XML handlers for one and the same XML namespace, such a handler is, before its execution, asked whether the currently chosen output format is supported or not. Therefore the `supportsRenderer` method is called with the current renderer object and the XML handler can decide itself.
- Finally the `handleXML` method is responsibe for the processing of the corresponding XML data. Therefore, the XML handler receives a `RendererContext` object, containing the current `Renderer` and various additional information, depending on the current output format. In addition to this context information, an `org.w3c.dom.Document` object, representing the custom XML structure, is also given to the XML handler.

5.2.2 Rendering to PDF

This work is primarily focused on the generation of PDF files containing rich 3D content. Therefore, the Apache FOP Java packages `org.apache.fop.render.pdf` and `org.apache.fop.pdf` are of great relevance, because they encapsulate all PDF related classes. In contrast to the `org.apache.fop.render.pdf` package, which contains all renderer-related classes, the package `org.apache.fop.pdf` covers various PDF objects, such as the basic types and inherited implementations.

Since the `FO3DPDFHandler` is just supporting PDF as a render target, the `handleXML` method will be called with a `RendererContext` that contains a reference to the `PDFDocument` object and the `PDFRenderer` itself. As the name suggests, the `PDFDocument` class is the object-oriented representation of the resulting PDF document and provides various public methods like `addObject`, which allows the appending of additional PDF objects, such as 3D annotations and 3D streams.

Almost all Java equivalents to relevant PDF objects are inherited from the `PDFObject` base class. Since general PDF dictionary and stream classes are already available in Apache FOP, all 3D-related classes defined by the FO3D plugin are based on existing implementations. For example, the PDF *3D Stream Dictionary* is in the FO3D plugin represented by the class `PDF3DStream`, which extends the `PDFStream` class of Apache FOP. Figure 5.1 provides an overview of 3D-related PDF classes, complemented with the FO3D Apache FOP plugin.

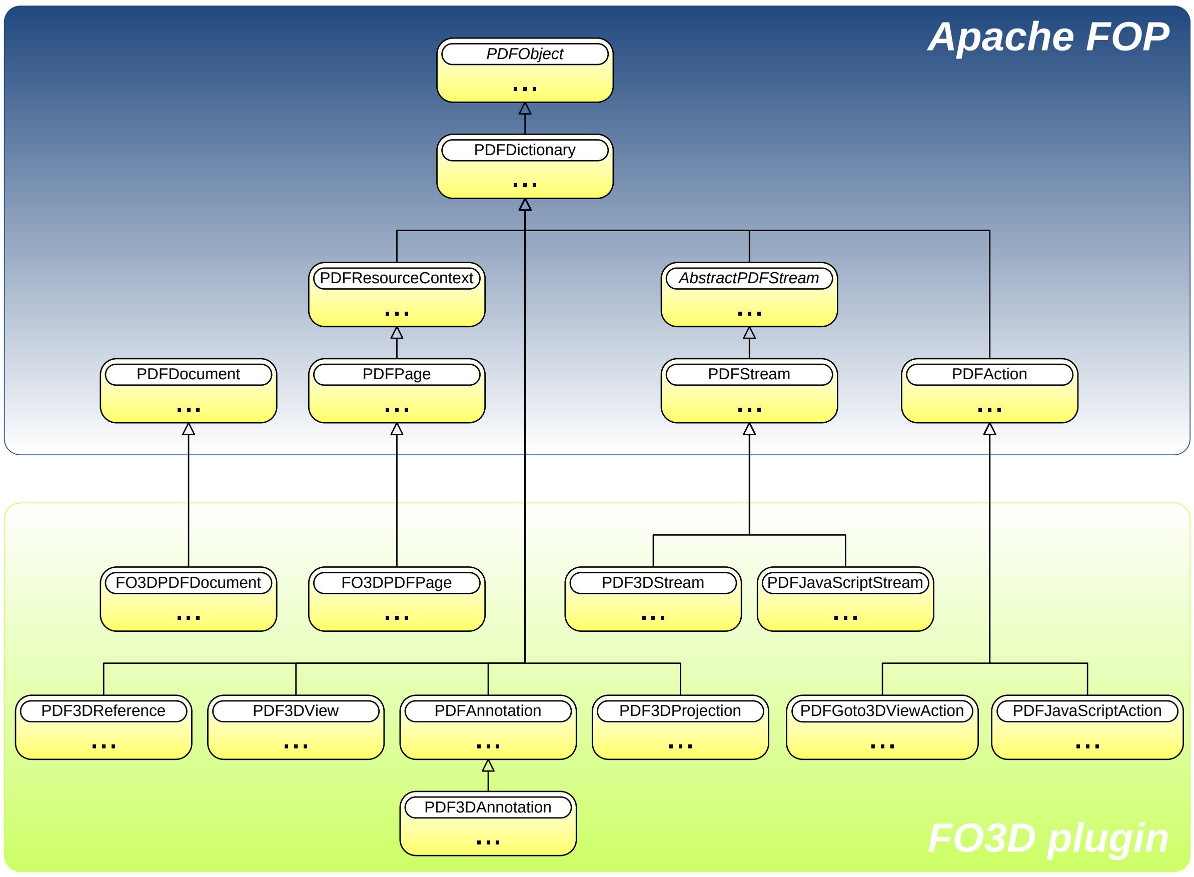


Figure 5.1: This class diagram shows a small subset of PDF-related Java classes defined by Apache FOP and extended 3D-related implementations introduced by the FO3D plugin.

All elements of the FO3D vocabulary described in Section 4.3 can now be represented by corresponding Java classes. Therefore, the next steps are the parsing of the FO3D XML input, the instantiation of corresponding 3D-related Java classes and the integration into the `PDFDocument` object.

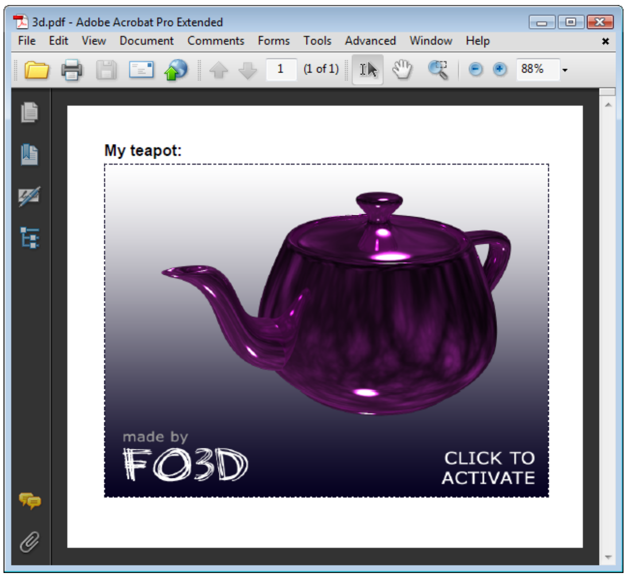
Since Apache FOP is designed for the rendering and integration of two-dimensional objects only, `XMLHandler` classes are just provided with sufficient support for that purpose. For the successful integration of FO3D, a deeper intervention is necessary and therefore the `PDFRenderer`, `PDFDocument` and `PDFPage` classes are extended with the similar named `FO3DPDFRenderer`, `FO3DPDFDocument` and `FO3DPDFPage` classes. Because the renderer class is responsible for the instantiation of the document and the page objects, all of the three extended implementations can be used if Apache FOP is adjusted to use the `FO3DPDFRenderer` instead of its own `PDFRenderer` class. This is simply done by registering the new renderer as described in Section 5.1. Since Apache FOP registers renderers according to their mimetype and the inherited class, by default, has the same mimetype as its parent, the `FO3DPDFRenderer` is as of now used for rendering PDF documents. With minor changes applied, the `FO3DPDFDocument` and `FO3DPDFPage` class are also used instead of their parent classes.

Now, having full control over the PDF rendering process, the `FO3DPDFHandler` just has to traverse the given FO3D DOM tree and add the corresponding stream or dictionary objects to the PDF document. Referring to the list of supported render targets in Apache FOP, it might be confusing that the supported PDF version is 1.4 and PDF is providing support for 3D artworks as of version 1.6. The FO3D plugin also eliminates this inconsistency and changes the version number to 1.7, because the FO3D vocabulary is mainly based on the PDF 1.7 specification [4].

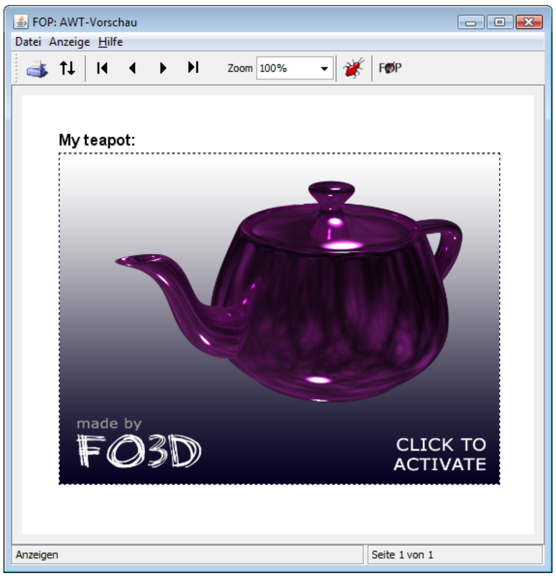
5.2.3 Non-PDF Output Formats

The `FO3DDefaultXMLHandler` class also implements the `XMLHandler` Java interface and provides additional support for rendering FO3D to several non-PDF output formats. Since the `FO3DPDFHandler` is strictly focused on PDF, the `FO3DDefaultXMLHandler` supports multiple output formats and uses the 2D graphics that can be defined with the `altsrc` attribute of the `fo3d:object-3d` element for an adequate two-dimensional representation of FO3D data. The following output formats are supported by the `FO3DDefaultXMLHandler` class:

- Printer Command Language (PCL)
- Advanced Function Presentation (AFP)
- Java2D/AWT (Java2D-based output formats: AWT viewer, direct print, PNG, TIFF, ...)
- PostScript


3d.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
1 (1 of 1) 88%
My teapot:
made by FO3D
CLICK TO ACTIVATE

(a) PDF with deactivated 3D annotation



(b) Apache FOP Java AWT viewer

Figure 5.2: Alternative graphics are used for multiple output formats such as PDF (a), in case the corresponding 3D artwork is deactivated, and also two-dimensional render targets like the Apache FOP Java AWT Viewer (b).

If the `altsrc` attribute is absent, which is possible since it is optional, the area that is defined by the `width` and the `height` attribute will remain blank. Figure 5.2 shows the visual result of such an alternative graphics for both, a deactivated 3D artwork embedded in a PDF document (a), and the two-dimensional preview of the document layout in the Apache FOP Java AWT Viewer (b).

Similar to PDF, other non strictly two-dimensional output formats could in theory also provide support for 3D content that is described with the FO3D vocabulary. For example, the Java AWT Viewer could be extended to use the Java 3D API[1] for showing 3D objects either, but since the FO3D vocabulary is closely related to the PDF output format, several elements and attributes may not be applicable in their current form.

5.2.4 JavaScript Support

XSL-FO is designed to describe static two-dimensional layouts and therefore Apache FOP does not provide any possibility to integrate things like JavaScript for PDF documents. Since Apache FOP is extendable, this feature can also be added in the same manner like the previously described 3D-related PDF objects. Therefore, FO3D introduces the `PDFJavaScriptStream` and the `PDFJavaScriptAction` classes, as shown in Figure 5.1. The `PDFJavaScriptStream` class represents the real JavaScript code, which can be defined by using the FO3D `script` and `doc-script` elements.

[1]`https://java3d.dev.java.net`

In case of 3D-related JavaScript code, a `PDFJavaScriptStream` is directly attached to an FO3D `PDF3DStream` object containing the 3D content. As described in Section 4.3.5, this connection is in PDF documents established by the *OnInstantiate* key of a 3D stream dictionary and a corresponding indirect reference to the JavaScript stream object.

Document-context JavaScript code is also represented by a `PDFJavaScriptStream` object, but cannot be directly linked to the catalog or page dictionary, because both require a *PDF Action Dictionary*, which will be executed in response to an occuring event like the opening of the document or page. Therefore, the FO3D plugin uses the `PDFJavaScriptAction` class. Referring to the PDF specification, this action dictionary can contain JavaScript code in two forms, as a direct string object or an indirect reference to a JavaScript stream object. Concerning document-level JavaScript, the `PDFJavaScriptStream` object is referred to from a `PDFJavaScriptAction` object, which itself is referenced by the corresponding catalog or page dictionary, depending on the chosen event type.

JavaScript Code Compression

Since JavaScript is a scripting language that is usually interpreted and not compiled, the source code often contains information that is not important for its execution, such as comments, line breaks, unnecessary brackets and even dead code. To save memory and interpretation time, these code fragments can be removed without causing any changes to the behavior of the code. Code compression is also often accompanied by code obfuscation, which provides at least little intellectual property protection. Apart from PDF documents, JavaScript is more heavily used in HTML web pages and since less code in this field means faster loading and also more users, several tools for compressing HTML, images and even JavaScript code have already been developed. Even though the underlying JavaScript API of a conforming PDF viewer application is totally different to those known from web browsers, it still remains JavaScript and the syntax rules are the same. Hence, many web-based JavaScript compressors can also be used for reducing the size of JavaScript code written for PDF documents.

Therefore the FO3D Apache FOP plugin supports the following two JavaScript compressors for reducing the size of both, 3D-related and document-context JavaScript code:

- Google Closure Compiler[2]
- Yahoo YUI Compressor[3]

Both JavaScript compressors are, similar to Apache FOP, written in the Java programming language and can be obtained in the form of a Java archive (JAR). To enable the JavaScript

[2]`http://code.google.com/closure/compiler/`
[3]`http://developer.yahoo.com/yui/compressor/`

```xml
<?xml version="1.0"?>
<fop version="1.0">
  <renderers>

    <renderer mime="application/pdf">

      <!-- FO3D XML Handler Settings -->
      <xml-handler namespace="http://fo3d.sourceforge.net/ns">
        <!-- possible values: closure-compiler, yuicompressor, none -->
        <javascript-code-compression>closure-compiler</javascript-code-compression>
      </xml-handler>

    </renderer>

  </renderers>
</fop>
```

Listing 5.1: Exemplary use of a configuration file to customize the FO3D JavaScript compression in Apache FOP.

compression for `PDFJavaScriptStream` objects, the corresponding JAR-file has to be put in the classpath, like the FO3D plugin itself. Since the FO3D plugin is able to use both compressors dynamically, no extra configuration is necessary. In addition to the JavaScript compression enabled by adding one of these two JAR-files, the Closure Compiler also privides programmatic access to a web-based API, referred to as *Closure Compiler Service API*. The FO3D plugin can also be configured to use this remote service, but obviously the remote API is much slower than the local Java archive and at least requires a connection to the Internet. Listing 5.1 provides an example of how Apache FOP can be used with a configuration file to enable or disable a specific JavaScript compressor. A registered XML handler, such as the FO3D plugin, can then access the configuration settings in the given `RendererContext`.

Apart from any custom configuration, the FO3D plugin does by default use one of the described JavaScript compressors if it is available in the classpath. Otherwise, no JavaScript compression will be applied. If the FO3D plugin is explicitly configured to use the *Closure Compiler* but is not present in the classpath, the previously mentioned remote *Closure Compiler Service API* will be used instead.

5.2.5 FO3D Extensions

As described in Section 4.3.7, FO3D proposes the `extensions` element to provide additional support for a more flexible way of generating rich 3D visualizations that are embedded in PDF documents. Similar to the FO3D plugin itself, custom-made FO3D extension handlers can be registered in the same manner as an additional renderer or XML handler is registered to Apache FOP.

The Java package `net.sourceforge.fo3d.extensions` contains a manageable interface called `ExtensionXMLElementHandler`, which has to be implemented by conforming FO3D extensions.

```
public interface ExtensionXMLElementHandler {

  /*
   * Handles a custom FO3D extensions element.
   */
  public void handleElement(RendererContext context, Element el, PDF3DAnnotation
      annot) throws Exception;

  /*
   * Returns the name of the extension element that
   * is supported by the current handler.
   */
  public String getSupportedElementName();

}
```

Listing 5.2: Java interface that has to be implemented by custom-made FO3D extension handlers

As shown in Listing 5.2, this interface just requires the implementation of two different methods. The `getSupportedElementName` method is just responsible for returning the supported element name, which is used during the registration of the current extension handler. Similar to the extension mechanism of Apache FOP, custom extension-element handlers are registered to the FO3D plugin by storing the fully qualified name of the handler class in the file:

"*/META-INF/services/net.sourceforge.fo3d.extensions.ExtensionXMLElementHandler*"

Afterwards a JAR-file has to be created and put in the classpath. If so, the FO3D plugin automatically registers such extension-element handlers during the bootstrap initialization.

The second and more interesting method in Listing 5.2 is called `handleElement` and it is responsible for the processing of the extension-elements that the corresponding handler is registered for. The FO3D plugin delivers the following three parameters to the custom extension handler:

The `RendererContext` represents the same object as Apache FOP hands over to the FO3D plugin. Hence, such a custom FO3D extension handler also has access to the PDF renderer and document objects, as previously described for the FO3D XML handler class itself.

The XML element is given to this handler in the form of an `org.w3c.dom.Element` object, containing all attributes and descendants of the extension element that shall be processed.

The PDF3DAnnotation object provides an FO3D extension handler with all necessary information about the current 3D artwork and enables the adding of additional views, resources and JavaScript code.

5.3 Extending the `fo:basic-link` element

In addition to the previously described Java classes and its PDF equivalents, the *Go-To-3D-View Action* was also introduced with the support of 3D artworks in PDF documents. This action dictionary is used to switch to a specific predefined view of a 3D annotation. This feature is not by default supported, neither in XSL-FO nor in Apache FOP, but for a more elegant way of combining textual information and 3D content, its presence is highly desirable. Refering to XSL-FO, the described action has a similar behavior as a hyperlink and therefore the `fo:basic-link` and its `external-destination` attribute are of great interest. This attribute normally just accepts an URI, but the FO3D Apache FOP plugin provides additional support for the definition of such a *Go-To-3D-View Action*. Therefore, the value of the `external-destination` attribute can also have the following syntax:

```
external-destination="3dview:<AnnotationName>:<ViewName>"
```

The "3dview" keyword, the name of the corresponding 3D annotation and the name of the selected 3D view, are separated by a colon, which implies that no colons should be used in the `name` attributes of the `fo3d:object-3d` and the `fo3d:view` element.

In the same manner, a second extension to the `external-destination` allows the execution of JavaScript code in response to a *click* event on the defined area. Equal to the JavaScript definition in HTML, done by the `href` attribute of the anchor tag, the FO3D plugin enables the very same syntax to be used in the `external-destination` attribute of the `fo:basic-link` element. After the "javascript" keyword, followed by a colon, the real JavaScript code can directly be inserted into the same attribute that looks as follows:

```
external-destination="javascript:<AcrobatJavaScriptCode>"
```

Examples of both, the activation of a predefined view in a specific 3D annotation and the execution of PDF document-context JavaScript code, in response to a click event that is triggered by a certain area, are shown in Listing 5.3.

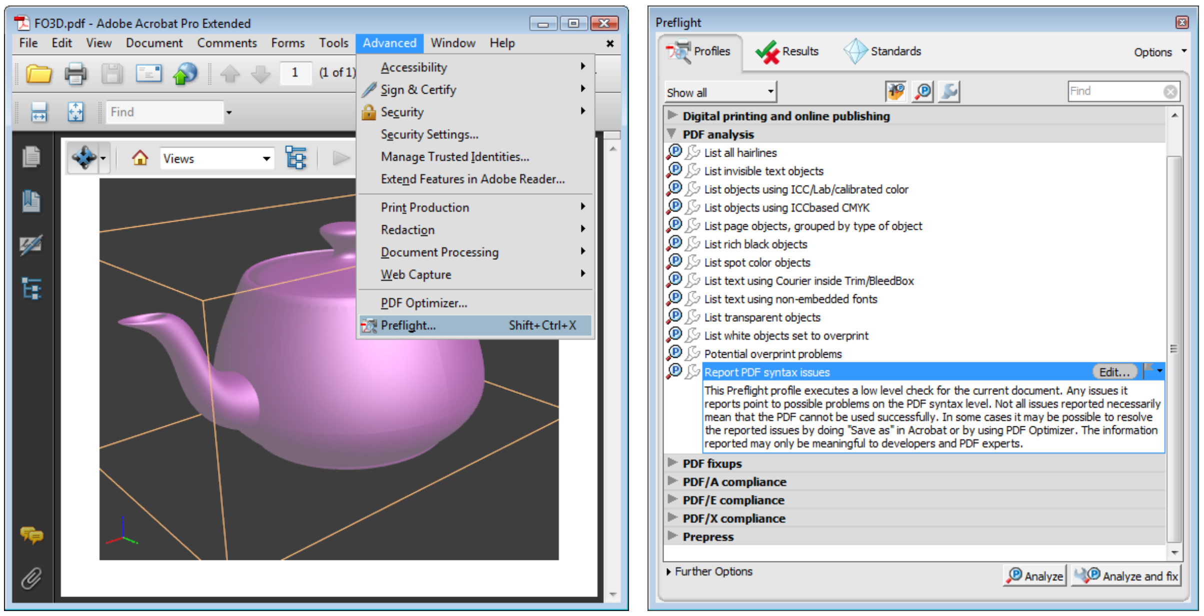
```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:fo3d="http://fo3d.
    sourceforge.net/ns">
  ...
    <fo:block>
      <fo:instream-foreign-object>
        <fo3d:object-3d src="./model.u3d" name="myAnnot" width="10cm" height="10cm"
            activation="page">
          <fo3d:views>
            <fo3d:view title="My special view" name="myView"/>
          </fo3d:views>
          <fo3d:scripts>
            <fo3d:script src="./3d_animation.js"/>
          </fo3d:scripts>
        </fo3d:object-3d>
      </fo:instream-foreign-object>
    </fo:block>

    <fo:block>
      <fo:basic-link external-destination="3dview:myAnnot:myView">
        Jump to the view!
      </fo:basic-link>
    </fo:block>

    <fo:block>
      <fo:basic-link external-destination="javascript:getAnnots3D(0)[0].context3D.
          startAnim();">
        Start 3D animation!
      </fo:basic-link>
    </fo:block>
  ...
</fo:root>
```

Listing 5.3: Exemplary usage of FO3D *3dview* and *javascript* actions.

5.4 PDF Syntax Validation

During the implementation of this FO3D prototype plugin for Apache FOP, the correct syntax of the resulting PDF documents containing 3D content has been validated by the *Preflight* tool that comes with Adobe Acrobat 9 Pro Extended. Apart from many useful scripts and the visualization of the internal PDF object tree, the correct syntax of a PDF document can be validated in the *PDF Analysis* section of the Preflight dialog, shown in Figure 5.3.

In addition to a correct PDF syntax, several special PDF standards have been developed and can also be validated with the Preflight tool. Although it would be desirable to support standards like PDF/X, facilitating graphics exchange or PDF/E, which is specifically tailored to engineering workflows, the mentioned validation routines will fail for the generated PDF documents. This is because Apache FOP is generally not supporting the generation of PDF documents that are compliant with one of these PDF sub-standards. Possibly, such a standard compliance may be introduced in future releases of Apache FOP and the FO3D plugin, respectively.


FO3D.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
Find
Views
Accessibility
Sign & Certify
Security
Security Settings...
Manage Trusted Identities...
Extend Features in Adobe Reader...
Print Production
Redaction
Document Processing
Web Capture
PDF Optimizer...
Preflight... Shift+Ctrl+X
Preflight
Profiles
Results
Standards
Options
Show all
Find
Digital printing and online publishing
PDF analysis
List all hairlines
List invisible text objects
List objects using ICC/Lab/calibrated color
List objects using ICCbased CMYK
List page objects, grouped by type of object
List rich black objects
List spot color objects
List text using Courier inside Trim/BleedBox
List text using non-embedded fonts
List transparent objects
List white objects set to overprint
Potential overprint problems
Report PDF syntax issues
Edit...
PDF fixups
PDF/A compliance
PDF/E compliance
PDF/X compliance
Prepress
Further Options
Analyze
Analyze and fix

This Preflight profile executes a low level check for the current document. Any issues it reports point to possible problems on the PDF syntax level. Not all issues reported necessarily mean that the PDF cannot be used successfully. In some cases it may be possible to resolve the reported issues by doing "Save as" in Acrobat or by using PDF Optimizer. The information reported may only be meaningful to developers and PDF experts.

Figure 5.3: Syntax validation of the generated PDF document, by using Acrobat Preflight.

Chapter 6

Use Cases

Referring to the proposed FO3D vocabulary and the corresponding prototype implementation based on Apache FOP, this chapter provides several examples for combining 3D content and related metadata. The resulting PDF documents have been tested with Acrobat 9 Pro Extended and the correctness of their syntax has been validated, as described in Section 5.4.

6.1 Use Case 1: Annotating 3D Objects

Based on the idea of adding textual information to certain locations in 3D space, this example shows how the workflow, presented in [43], could be realized and improved by using the FO3D vocabulary and the corresponding Apache FOP plugin.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<models>
  <model src="models/Laurana.u3d">
    <annotations color="#0F0" scale="0.4">
      <link url="http://en.wikipedia.org/wiki/Ear" pos="-4,-160,-3"/>
      <link url="http://fo3d.sourceforge.net" pos="8,-158,-2"/>
    </annotations>
  </model>

  <model src="models/teapot.u3d">
    <annotations>
      <link url="http://en.wikipedia.org/wiki/Utah_teapot" pos="0,0,14.64" scale="
          0.043" color="#FF0"/>
      <link url="http://en.wikipedia.org/wiki/Handle" pos="-10.74,0,7.79" scale="
          0.057" color="#F00"/>
    </annotations>
  </model>
</models>
```

Listing 6.1: XML input data for annotating the related 3D content.

For example, 3D content and related annotations may be encoded in an XML document, as shown in Listing 6.1. This XML document contains multiple 3D models, each enriched by an arbitrary number of related information that is referred to by an URL. Each `annotations`

```
 1 ...
 2 <fo:block break-before="page">
 3   <fo:instream-foreign-object content-width="scale-to-fit" width="100%">
 4     <object-3d src="models/teapot.u3d" height="9cm" width="9cm" name="annot02" xmlns
         ="http://fo3d.sourceforge.net/ns">
 5       <resources>
 6         <resource src="sphere.u3d" name="ball"/>
 7       </resources>
 8       <scripts>
 9         <script src="annotations.js">
10           add3DLink('ball', new Vector3(0,0,14.64), 'http://en.wikipedia.org/wiki/
                 Utah_teapot', 0.043, new Color(1,1,0));
11           add3DLink('ball', new Vector3(-10.74,0,7.79), 'http://en.wikipedia.org/wiki
                 /Handle', 0.057, new Color(1,0,0));
12         </script>
13       </scripts>
14     </object-3d>
15   </fo:instream-foreign-object>
16 </fo:block>
17
18 <fo:block margin-top="3mm" text-align="center" color="blue">
19   <fo:inline color="#000">Annotations:</fo:inline>
20   <fo:basic-link external-destination="javascript:show3DLinks('annot02');">
21     Show
22   </fo:basic-link>
23   <fo:basic-link external-destination="javascript:hide3DLinks('annot02');">
24     Hide
25   </fo:basic-link>
26   <fo:basic-link external-destination="javascript:toggle3DLinks('annot02');">
27     Toggle
28   </fo:basic-link>
29 </fo:block>
30 ...
```

Listing 6.2: Resulting FO document after transforming the XML input shown in Listing 6.1.

element, like the one shown in line 4, can define default values for the scale and the color of the contained link elements. The `url` attribute of such a `link` element specifies the URL it is referring to, its position in 3D space and optionally the size, respectively the color of the resulting clickable sphere that is used to display the link annotations in the 3D scene.

After applying an XSL transformation, the resulting FO document, especially the FO3D part of the "teapot" model, is shown in Listing 6.2. Each 3D model is displayed on a separate page, which is ensured by the `fo:break-before` attribute, such as shown in line 2. Furthermore, every `fo3d:object-3d` element gets a unique name, so that the resulting PDF 3D annotation dictionary can later be accessed by Acrobat JavaScript code.

In addition to the basic 3D content, the sphere model representing the hyperlinks in 3D space is added to every 3D artwork by using the `fo3d:resource` element. Since such resources can be instantiated by JavaScript as often as needed, the sphere model just has to be added once to every 3D stream.

It is assumed that the JavaScript code loaded from the external file "annotations.js" is implementing the required `add3DLink` function, shown in line 10 and 11. The `add3DLink` JavaScript function itself is presented in Listing 6.3. This function is responsible for loading the sphere by

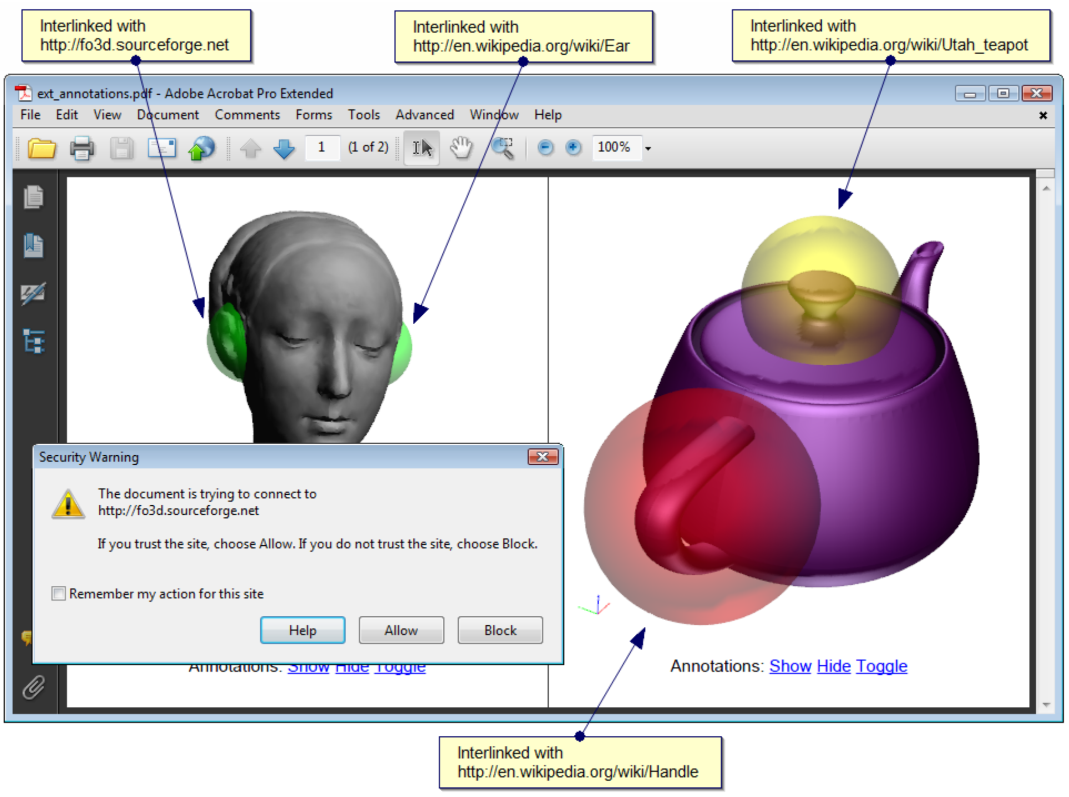
```javascript
function add3DLink(resName, position, uri, scale, color) {
  if (typeof(resName) !== "undefined" && resName &&
      typeof(position) !== "undefined" && position) {
    var res = new Resource("pdf://" + resName);
    if (res.type === Resource.TYPE_MODEL) {
      var newNode = addModelSafely(res);
      if (newNode) {
        newNode.renderMode = scene.RENDER_MODE_TRANSPARENT;
        if (typeof(color) !== "undefined" && color) {
          newNode.material.ambientColor.set(color);
          newNode.material.diffuseColor.set(color);
          newNode.material.specularColor.set(color);
        }
        if (typeof(scale) === "number") {
          newNode.transform.scaleInPlace(scale, scale, scale)
        }
        newNode.transform.translateInPlace(position);

        // Add event handler for new 3D link annotation
        if (typeof(uri) === "string" && uri) {
          var dest = uri;
          var eh = new MouseEventHandler();
          eh.onMouseDown = true;
          eh.reportAllTargets = false;
          eh.target = newNode;
          eh.onEvent = function(evt) {
            if (!evt.ctrlKeyDown) {
                return;
            }
            // call document over globa host variable
            if (typeof(host.app) !== "undefined") {
              host.app.launchURL(dest);
            }
          };
          runtime.addEventHandler(eh);
        }
        return true;
      }
    }
  }
  return false;
}
```

Listing 6.3: The JavaScript function `add3DLink` is used to load and initialize a 3D sphere object from the list of appended resources. The contained function `addModelSafely` is NOT part of the 3D JavaScript API, but used as a workaround which is explained in more detail in Section 6.4.2.

its resource name, attaching a click handler for redirecting to the given URL and placing the colored and scaled sphere at its designated position in the 3D scene. All these tasks can be done, by using the JavaScript functions from the PDF 3D JavaScript API [2].

For convenience, it is assumed that the three document-context JavaScript functions shown in line 20, 23 and 26 of Listing 6.2 and called `show3DLinks`, `hide3DLinks` and `toggle3DLinks`, have been previously defined with the `fo3d:doc-script` element of the "laurana" model. These functions are used for toggling the visibility of the hyperlinks contained in 3D space and they are added to the FO document by `fo:basic-link` elements.

After the FO rendering process of Apache FOP in co-operation with the FO3D plugin, a PDF document with two pages was generated. Both PDF pages are presented in Figure 6.1, which


Interlinked with http://fo3d.sourceforge.net
Interlinked with http://en.wikipedia.org/wiki/Ear
Interlinked with http://en.wikipedia.org/wiki/Utah_teapot
ext_annotations.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
1 (1 of 2) 100%
Security Warning
Remember my action for this site
Help Allow Block
Annotations: Show Hide Toggle
Interlinked with http://en.wikipedia.org/wiki/Handle

The document is trying to connect to
http://fo3d.sourceforge.net

If you trust the site, choose Allow. If you do not trust the site, choose Block.

Figure 6.1: 3D content and linked web resources, represented by colored spheres of different size. By clicking onto such a sphere, the linked web page will be opened in a browser window. Similar to other embedded hyperlinks in PDF documents, the user may previously be asked for permission.

also shows the security warning which might be displayed when clicking onto such a 3D hyperlink sphere.

Summarizing, the following listing outlines several advantages of the FO3D approach, in contrast to the related implementation presented by Strobl et al. [43]:

- FO3D is exclusively based on open standards, independent of proprietary software like Adobe Acrobat and the corresponding SDK. Furthermore, Apache FOP is an open-source project and can, in contrast to the commercial Adobe Acrobat, be obtained for free.
- The FO3D extension can be used in a web service solution, because Apache FOP can by default be used in a servlet environment.
- Both, U3D and PRC can be used as file formats for representing the 3D content.
- The basic 3D content remains untouched, which means that it could be extracted from the corresponding PDF 3D stream dictionary without containing any additional geometry, such as these hyperlink spheres.
- Additional geometry, such as the sphere model, just needs to be added once. Nevertheless, it can be instantiated multiple times.

- A JavaScript callback, launching the specified URL in response to a click event, is directly attached to the link sphere, which makes the definition and the parsing of a global URL-array obsolete. Apart from this matter of taste, the presented FO3D approach proposes the use of the `launchURL` function, which is specifically tailored to the opening of an URL in a browser window. The `getURL` function that has been used in [43], might load the requested URL directly within the PDF viewer application and not in a web browser window.

6.2 Use Case 2: 3D Text-Sprites

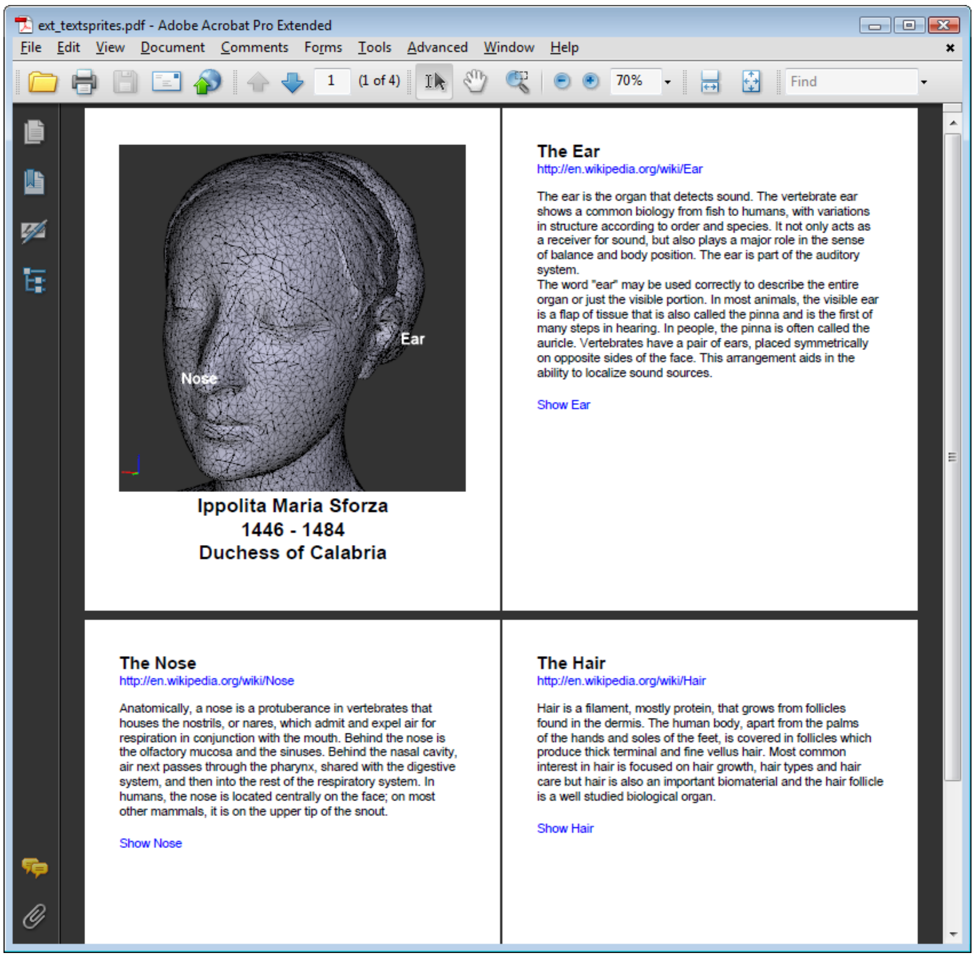
The main goal of this example is to show the definition of custom 3D views, as well as an exemplary implementation of a custom `fo3d:extensions` element, called `textsprite`. In contrast to the example shown in Section 6.1, where spheres represented additional information in 3D space, the `textsprite` extension is used to directly add human readable text in 3D space. The resulting PDF document shown in Figure 6.2, contains a 3D artwork on the first page and three more pages with additional textual information on specific parts of 3D content.

The used 3D model shows the bust of Ippolita Maria Sforza (1446-1484), Duchess of Calabria that was built around 1473 by Francesco Laurana (ca. 1420-1502), an Italian architect originally from Croatia. The corresponding U3D file has been taken from the 3D mesh processing software MeshLab, Version 1.2.2, which contains this file as a demonstration of the recently added U3D export functionality [48].

Starting from Version 9, Adobe Acrobat does provide support for real 3D comments, but the required PDF dictionaries are not part of the PDF 1.7 ISO specification [4]. 3D comments are defined as an extension to this specification, referred to as PDF 1.7 Adobe Extension Level 3, which is currently not fully supported FO3D. One major drawback of these 3D comments, already denoted by Strobl et al. [43], is that they provide no JavaScript event handlers, which means that it is not possible to use such 3D comments as hyperlinks to external web pages or internal destinations in the same PDF documents.

Nevertheless, FO3D can add human readable text to 3D content by appending two-dimensional images, which contain the rendered textual information, as resources to the 3D stream and loading these images by JavaScript as the texture of a rectangular area in 3D space. Such images are in computer graphics and the following description referred to as *sprites* or *textsprites*, respectively. It is not necessary to add additional geometry for the rectangular areas, because they can be dynamically created by using the `createSquareMesh` JavaScript function of the 3D API.

The four PDF pages presented in Figure 6.2, have been generated by using Aapche FOP, the FO3D plugin and a custom handler for the mentioned `textsprite` extension element.


ext_textsprites.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
1 (1 of 4) 70% Find
Ear
Nose

Ippolita Maria Sforza
1446 - 1484
Duchess of Calabria

The Ear
http://en.wikipedia.org/wiki/Ear

The ear is the organ that detects sound. The vertebrate ear shows a common biology from fish to humans, with variations in structure according to order and species. It not only acts as a receiver for sound, but also plays a major role in the sense of balance and body position. The ear is part of the auditory system.
The word "ear" may be used correctly to describe the entire organ or just the visible portion. In most animals, the visible ear is a flap of tissue that is also called the pinna and is the first of many steps in hearing. In people, the pinna is often called the auricle. Vertebrates have a pair of ears, placed symmetrically on opposite sides of the face. This arrangement aids in the ability to localize sound sources.

Show Ear

The Nose
http://en.wikipedia.org/wiki/Nose

Anatomically, a nose is a protuberance in vertebrates that houses the nostrils, or nares, which admit and expel air for respiration in conjunction with the mouth. Behind the nose is the olfactory mucosa and the sinuses. Behind the nasal cavity, air next passes through the pharynx, shared with the digestive system, and then into the rest of the respiratory system. In humans, the nose is located centrally on the face; on most other mammals, it is on the upper tip of the snout.

Show Nose

The Hair
http://en.wikipedia.org/wiki/Hair

Hair is a filament, mostly protein, that grows from follicles found in the dermis. The human body, apart from the palms of the hands and soles of the feet, is covered in follicles which produce thick terminal and fine vellus hair. Most common interest in hair is focused on hair growth, hair types and hair care but hair is also an important biomaterial and the hair follicle is a well studied biological organ.

Show Hair

Figure 6.2: This example shows a 3D object on the first page, the bust of Ippolita Maria Sforza made by Francesco Laurana, and bidirectionally interlinked related information on subsequent pages.

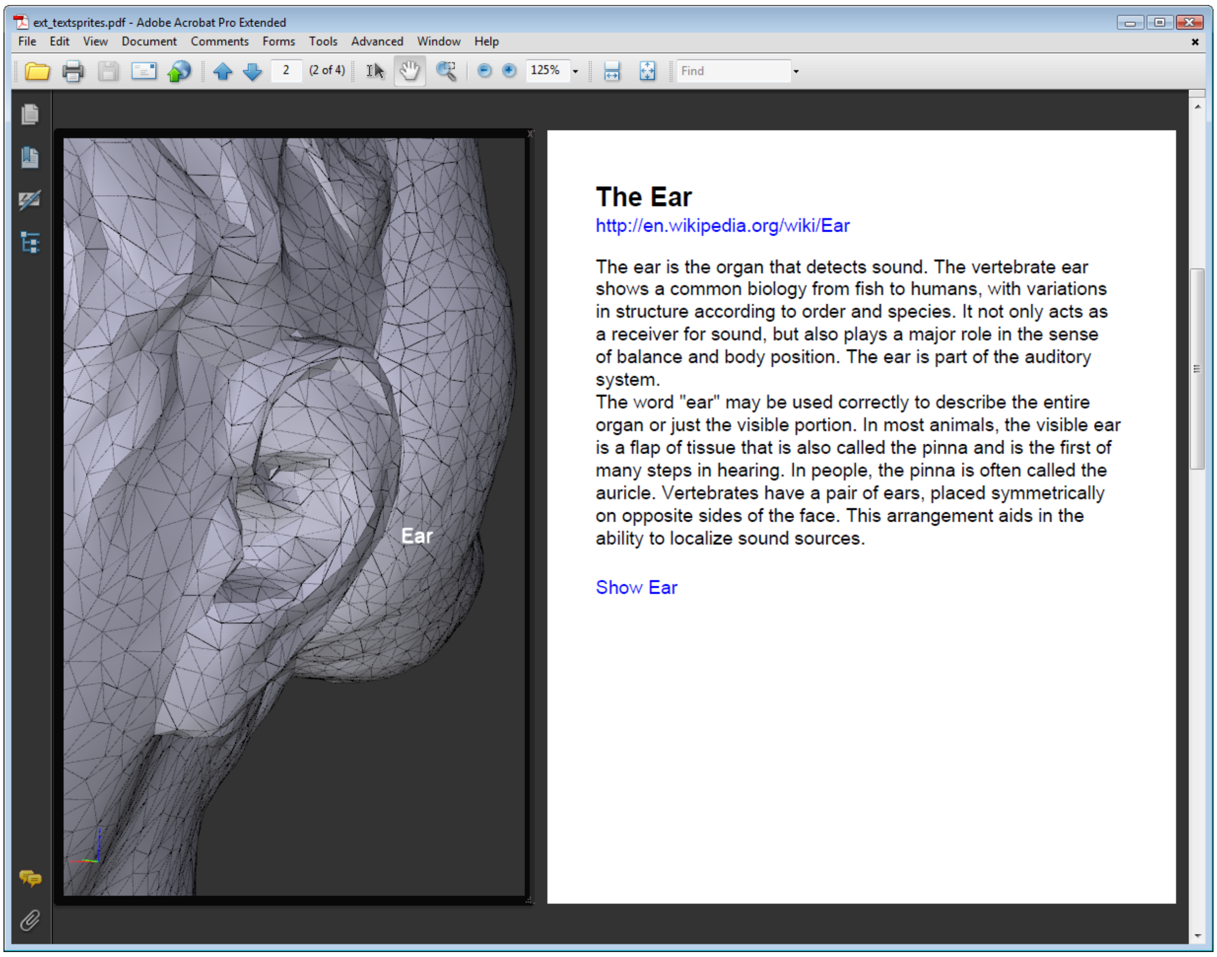
This extension handler is nothing more than a single Java class that implements the FO3D `ExtensionXMLElementHandler` interface and tells the FO3D Apache FOP plugin to use this handler `textsprite` elements.

A small extract from the FO document that is used to generate the final PDF file is presented in Listing 6.4. Apart from the included text itself, the `textsprite` element supports the two attributes `at` and `internal-destination`. The `at` attribute represents a point in 3D space, where the rectangular area and therefore the readable text shall be displayed. The `internal-destination` attribute is used to define the name of a PDF-internal destination, referred to as *Named Destination*.

To create such a named destination, the standard extension namespace of Apache FOP defines the `fox:destination` element. As shown in line 28 of Listing 6.4, the `internal-destination` attribute of the `fox:destination` element takes the `id` of an FO element, in that case of the `fo:block` element in line 26, and creates a named destination called “ear” that in the final document refers to the appropriate position of this FO element.

```
...

<fo:instream-foreign-object width="100%" content-width="scale-to-fit">
  <object-3d xmlns="http://fo3d.sourceforge.net/ns" name="model" src="laurana.u3d"
      width="5cm" height="5cm" activation="visibility" deactivation="user" after-
      deactivation="live">
    <views>
      <view title="Ear" name="ear" default="true">
        <camera position="-304,-1512,-31" target="-69,-1589,-34" up="0,0,1" />
      </view>
      <view title="Hair" name="hair">
        <camera position="-259,-1920,117" target="-15,-1704,22" up="0,0,1" />
      </view>
      <view title="Nose" name="nose">
        <camera position="-125,-1489,-31" target="10,-1496,-34" up="0,0,1" />
      </view>
    </views>
    <extensions>
      <textsprite at="-80,-1590,-35" internal-destination="ear">Ear</textsprite>
      <textsprite at="20,-1750,10" internal-destination="hair">Hair</textsprite>
      <textsprite at="12,-1470,-50" internal-destination="nose">Nose</textsprite>
    </extensions>
  </object-3d>
</fo:instream-foreign-object>

...

<fo:block break-before="page" id="ear">
  <fo:block font-size="14pt" font-weight="bold"> The Ear </fo:block>
  <fox:destination internal-destination="ear" />

  <fo:block color="blue">
    <fo:basic-link external-destination="http://en.wikipedia.org/wiki/Ear">http://en.
        wikipedia.org/wiki/Ear</fo:basic-link>
  </fo:block>
  <fo:block margin-top="10pt">
    The ear is the organ that detects sound ...
  </fo:block>

  <fo:block color="blue" margin-top="5mm">
    <fo:basic-link external-destination="3dview:model:ear">Show Ear</fo:basic-link>
  </fo:block>
</fo:block>

...
```

Listing 6.4: FO(3D) document containing predefined views and custom `textsprite` extension elements for adding human readable information to certain locations in 3D space.


ext_textsprites.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
2 (2 of 4) 125% Find
Ear

The Ear

http://en.wikipedia.org/wiki/Ear

The ear is the organ that detects sound. The vertebrate ear shows a common biology from fish to humans, with variations in structure according to order and species. It not only acts as a receiver for sound, but also plays a major role in the sense of balance and body position. The ear is part of the auditory system.
The word "ear" may be used correctly to describe the entire organ or just the visible portion. In most animals, the visible ear is a flap of tissue that is also called the pinna and is the first of many steps in hearing. In people, the pinna is often called the auricle. Vertebrates have a pair of ears, placed symmetrically on opposite sides of the face. This arrangement aids in the ability to localize sound sources.

Show Ear

Figure 6.3: This example shows the 3D artwork in a floating window, always visible beside the pages of the PDF document.

The custom `textsprite` handler on-the-fly generates the images containing the given text, by using Java classes from the `java.awt.*` packages, such as `BufferedImage` or `TextLayout` and appends them as a resource to the current 3D stream object. Furthermore, it injects JavaScript code that loads all generated images as textures of a rectangular area and dynamically adjusts their orientation, so that they are always readable in the 2D viewport.

If the `internal-destination` attribute of the `textsprite` element is set, a click event handler, redirecting to the corresponding named destination, is attached too. The Acrobat JavaScript API therefore provides the `gotoNamedDest` function.

Starting from line 5, Listing 6.4 also shows three predefined views, each defining a different camera position and orientation. This facilitates the exploration of specific parts in the embedded 3D scene. Referring to Adobe Acrobat, such predefined views can be selected in the drop-down list box included in the interactive toolbar that is usually displayed above the 3D artwork itself.

The second possibility to select such a predefined view is by clicking onto a *Go-To-3D-View* action that has been added to the document by using the `external-destination` attribute of `fo:basic-link` elements.

To sum up the features of the presented use case, it can be stated that the resulting PDF document does not only contain 3D content and related textual information for specific parts of the 3D object, e.g. *The Nose*, *The Ear* and *The Hair*, but does also provide an interactive connection between both. Additional information in the document can be accessed by clicking on a textsprite in 3D space and inversely, after having selected a predefined view by a hyperlink that is directly embedded in the content of the PDF document, the 3D artwork can show specific parts in greater detail.

Since Adobe Acrobat does, on executing a *Go-To-3D-View Action*, admittedly change the view of the 3D annotation itself, but does not jump to the corresponding page, this might be annoying. Therefore, Figure 6.3 shows an alternative method for placing the 3D artwork beside the pages of the PDF document. This enables the user to synchronously read the textual information and explore corresponding parts in 3D space. Such an alternative viewport is, in Adobe Acrobat, referred to as *Floating Window* and can be enabled in the context menu of the 3D artwork.

Finally, it has to be mentioned that the generated images are not optimal for adding textual information in 3D space, but this example was primarily designed to illustrate how a two-dimensional document and a 3D artwork can be combined interactively. In addition to that, the power of using a custom FO3D extension handler has also been demonstrated successfully.

6.3 Use Case 3: PGN Chess Game Visualization

Due to visualization, exchange, replay and study issues, chess games are often stored in computer-processible formats. An extremely popular format implemented by almost every chess program available is the Portable Game Notation (PGN), developed by Steven J. Edwards around 1993 [24]. Its big success may be referrable to the human readability and the easy parsing, respectively generation by computer programs.

```
[Event "New York Rosenwald"]
[Site "New York"]
[Date "1956.10.07"]
[White "Byrne,Donald"]
[Black "Fischer,Robert James"]
[Result "0-1"]
[Eco "D97"]

1.Nf3 Nf6 2.c4 g6 3.Nc3 Bg7 4.d4 0-0 5.Bf4 d5 6.Qb3 dxc4 7.Qxc4 c6 8.e4 Nbd7
9.Rd1 Nb6 10.Qc5 Bg4 11.Bg5 Na4 12.Qa3 Nxc3 13.bxc3 Nxe4 14.Bxe7 Qb6 15.Bc4 Nxc3
16.Bc5 Rfe8+ 17.Kf1 Be6 18.Bxb6 Bxc4+ 19.Kg1 Ne2+ 20.Kf1 Nxd4+ 21.Kg1 Ne2+
22.Kf1 Nc3+ 23.Kg1 axb6 24.Qb4 Ra4 25.Qxb6 Nxd1 26.h3 Rxa2 27.Kh2 Nxf2 28.Re1 Rxe1
29.Qd8+ Bf8 30.Nxe1 Bd5 31.Nf3 Ne4 32.Qb8 b5 33.h4 h5 34.Ne5 Kg7 35.Kg1 Bc5+
36.Kf1 Ng3+ 37.Ke1 Bb4+ 38.Kd1 Bb3+ 39.Kc1 Ne2+ 40.Kb1 Nc3+ 41.Kc1 Rc2+ 0-1
```

Listing 6.5: PGN encoded chess game, nicknamed as "The Game of the Century", between Donald Byrne and the 13-year old Robert "Bobby" Fischer.

```
...
<fo:instream-foreign-object xmlns="http://fo3d.sourceforge.net/ns">
  <object-3d src="board.u3d" width="10cm" height="6cm" activation="visibility">
    <views>
      <view title="Above" light-mode="headlamp" default="true">
        <camera position="0,0,18" target="0,0,0" up="-1,0,0" />
      </view>
      <view title="White Player">
        <camera position="0,-16,10" target="0,0,0" up="0,0,1" />
      </view>
      <view title="Black Player">
        <camera position="0,16,10" target="0,0,0" up="0,0,1" />
      </view>
      <view title="Guest">
        <camera position="16,0,10" target="0,0,0" up="0,0,1" />
      </view>
    </views>
    <resources>
      <resource src="king.u3d" name="king"/>
      <resource src="queen.u3d" name="queen"/>
      <resource src="bishop.u3d" name="bishop"/>
      <resource src="knight.u3d" name="knight"/>
      <resource src="rook.u3d" name="rook"/>
      <resource src="pawn.u3d" name="pawn"/>
    </resources>
    <scripts>
      <script src="pgnchess.js"/>
    </scripts>
    <extensions>
      <!-- Loading, parsing PGN and generating JS annimations -->
      <pgn src="http://www.chessgames.com/pgn/fischer_byrne_1956.pgn?gid=1008361"/>
    <extensions>
  </object-3d>
</fo:instream-foreign-object>
...
```

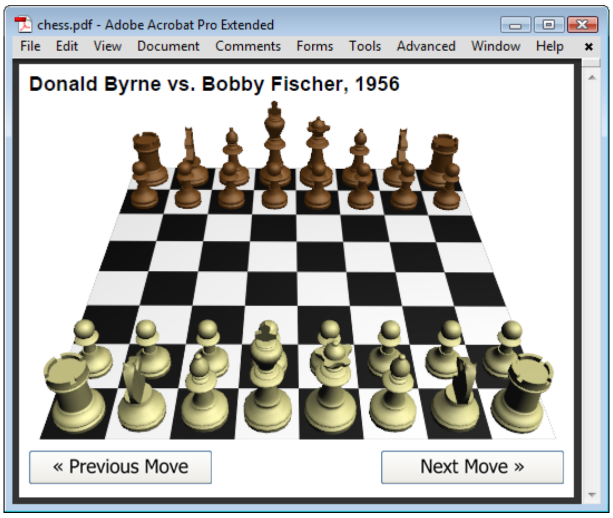
Listing 6.6: Exemplary definition of a 3D artwork representing an interactive visualization of a PGN based chess game.

An example of the PGN file format is shown in Listing 6.5, representing one of the most famous chess matches, often referred to as *The Game of the Century*. In this game, 13-year old Robert "Bobby" Fischer defeated his formidable opponent Donald Byrne, in October 17, 1956.

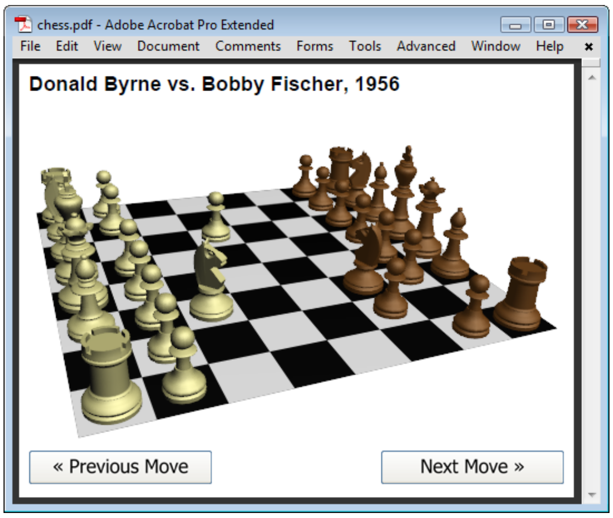
Commonly, a chess game is visualized by a two-dimensional representation of the board and the chessmen drawn at their current positions. This can be done for every single move and the final result would be a flat, static documentation of the complete game, which can be viewed like a flip-book. RenderX Inc., manufacturer of the XEP XSL-FO processor, therefore created an exemplary XSL-FO transformation[1] that turns a PGN/XML encoded chess game into a PDF document that contains every single position rendered in a separate two-dimensional view.

2D views of chessboards are quite nice, but adding the third dimension is even nicer. Therefore, various tools for displaying chess games in 3D have already been developed, but almost all tools require the installation of a proprietary software. In the following, an FO3D based solution for embedding such a chess visualizations in PDF documents is presented. An example of how the FO3D part of a corresponding FO document could look like, is shown in Listing 6.6.

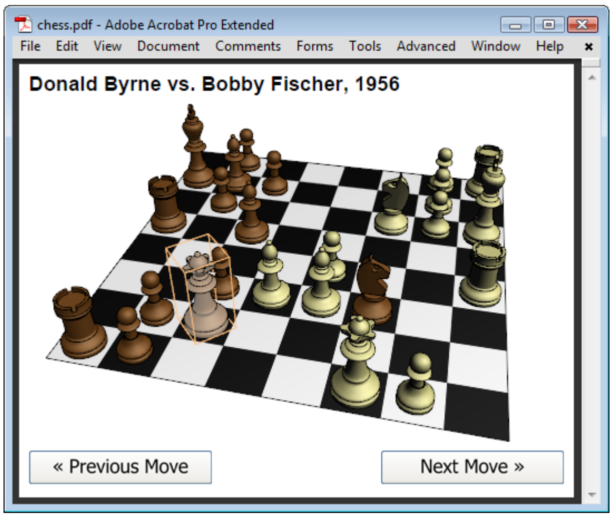
[1]http://www.renderx.com/chess.html


chess.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
Donald Byrne vs. Bobby Fischer, 1956
« Previous Move
Next Move »

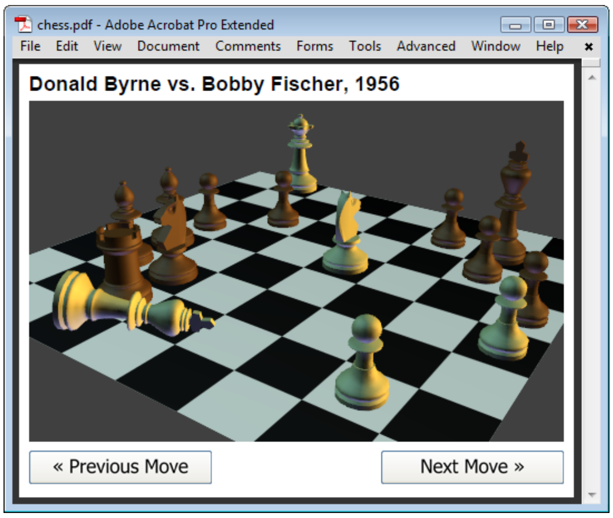
(a) Initial position


chess.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
Donald Byrne vs. Bobby Fischer, 1956
« Previous Move
Next Move »

(b) After 2 moves


chess.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
Donald Byrne vs. Bobby Fischer, 1956
« Previous Move
Next Move »

(c) Queen Sacrifice


chess.pdf - Adobe Acrobat Pro Extended
File Edit View Document Comments Forms Tools Advanced Window Help
Donald Byrne vs. Bobby Fischer, 1956
« Previous Move
Next Move »

(d) Checkmate

Figure 6.4: Various positions and different views of the historical chess game, in which Robert "Bobby" Fischer (black) defeated Donald Byrne (white) in 1956.

It is planned to support the real PGN format and not an XML copy, as it was done in the previously mentioned XSL-FO example from RenderX Incorporated. Therefore, a custom fO3D extension handler is needed, because otherwise the external PGN file cannot be loaded during the PDF generation process. For this purpose, the `pgn` extension element, supporting just one attribute called `src`, has been introduced. This handler just has to load the PGN file from the given URL and transform it into a JavaScript structure, which is appended to the current JavaScript stream object. For the parsing and the transformation of the given PGN encoded data, numerous software solutions written in Java or even JavaScript are freely available. All necessary JavaScript code for working with this data structure and initializing the 3D artwork is defined and loaded with the file "pgnchess.js", illustrated in line 27 of Listing 6.6. If the corresponding PDF 3D annotation becomes active now, the JavaScript code can enable a step-by-step animation of every single move of the PGN-encoded chess game. The resulting PDF document and different views and positions of the chess game are shown in Figure 6.4.

As previously mentioned, resources can be instantiated multiple times and therefore all chessman models are just added once. By means of JavaScript, every chessman model is loaded as often as needed, colored in black or white and positioned on the chessboard. Finally, predefined views enable the viewer to watch the chess game with the eyes of the black or white player, but it is also possible to navigate arbitrarily through the scene.

All in all, the described FO3D-based approach for visualizing chess games interactively in 3D space provides new perspectives and can facilitate the studying of chess openings or endgames. The major advantage in contrast to other 3D chess visualizations is that no extra software is needed. A conforming PDF viewer, like Adobe Reader, is fully sufficient, freely available and already installed on a great number of computers.

6.4 Known Issues

During the implementation of the FO3D Apache FOP plugin described in Chapter 5 and the development of the previously shown application examples, various problems and peculiarities occurred. The following subsections summarize the emerged issues, detected with the PDF viewer application Adobe Acrobat 9.3.1 Pro Extended on Microsoft Windows Vista Home Premium (32 bit).

6.4.1 JavaScript Function `gotoNamedDest` Crashes Acrobat

Under certain circumstances Adobe Acrobat crashes, if the JavaScript function for redirecting to a named destination called `gotoNamedDest` is directly executed in response to an attached click event on a certain 3D object. A quick and dirty workaround shown by the following JavaScript code, waits for a second and then redirects to the given named destination.

```
host.app.setTimeOut("this.gotoNamedDest('"+internalDestination+"');", 1000);
```

A closer look at the corresponding 3D artwork and more experiments with adequate JavaScript code identified the state of the 3D annotation as the cause for the program crash.

```
var eh = new MouseEventHandler();
eh.onMouseDown = true;
eh.target = scene.meshes.getByIndex(0);
eh.onEvent = function(evt) {
  host.gotoNamedDest('my-named-destination');
};
runtime.addEventHandler(eh);
```

Listing 6.7: JavaScript redirection to a named destination, in response to a click event on an object in 3D space.

It turned out that this error occurs if the named destination is located on a different page than the current 3D annotation. The redirection to another page fully deactivated the 3D annotation before the JavaScript callback, attached with a `MouseEventHandler`, has finished its work. This is because after leaving a page, the default state of a 3D annotation is *uninstantiated*. Setting the `after-deactivation` attribute of the `fo3d:object-3d` element to *instantiated* or even *live* enables the execution of the JavaScript code, illustrated in Listing 6.7, without any errors or program crashes.

6.4.2 3D JavaScript Function `addModel` Returns `null`

For adding a three-dimensional object that has been attached as a resource to the 3D stream dictionary, the 3D JavaScript API provides the `addModel` function. An example of how this function can be used to dynamically load such a 3D object is presented in Listing 6.8.

```
1 var res = new Resource("pdf://my-3d-object");
2 var newNode = scene.addModel(modelRes);
3
4 // the following line produces an error because newNode is null
5 newNode.transform.translateInPlace(new Vector3(28.3, 38.78, 0));
```

Listing 6.8: The 3D JavaScript function `addModel` returns `null` instead of the `Node` object, representing the added 3D model.

Resources, especially 3D objects and images, are loaded in JavaScript with the constructor of the `Resource` object given a valid resource name. In FO3D, the name of a resource can be specified with the `name` attribute of the corresponding `resource` element, as desribed in Section 4.3.6. After the successful instantiation of the Resource object shown in line 1 of Listing 6.8, the variable `res` represents the 3D object. Referring to the 3D JavaScript API, the `addModel` function adds the given `Resource` to the top level of the scene and returns a `Node` object representing the top-level `Mesh` object of the loaded model. Indeed, the `addModel` function adds the corresponding

3D content to the scene, but does NOT return the expected `Mesh` object, but `null` instead. Therefore, the JavaScript command for applying a translation as shown in line 5 of Listing 6.8 will cause an error, since the variable `newNode` is `null`. This problem has already been discussed in various related online forums, but has not yet been fixed by Adobe.

```
function addModelSafely(modelRes) {
  var oldCnt = scene.meshes.count;
  scene.addModel(modelRes);
  if (scene.meshes.count > oldCnt) {
    return scene.meshes.getByIndex(oldCnt);
  }
}
```

Listing 6.9: A custom-made workaround that corrects the misbehavior of the `addModel` function from the 3D JavaScript API.

All presented examples in this work therefore used the custom-made function `addModelSavely` instead, which provides a straightforward workaround, as illustrated in Listing 6.9. This function stores the number of available nodes from the list of meshes before adding the new 3D resource to the scene. After the `addModel` function has been called and the new number of available nodes is greater than the one that is stored, the last entry of this list is returned. In case the 3D content resource contains more than one top-level node, this function has to be adjusted.

6.4.3 Undefined `host` Variable in the 3D JavaScript API

Several JavaScript tests for accessing the PDF document JavaScript object have failed during the implementation of the FO3D plugin. A closer inspection using the JavaScript console of Adobe Acrobat assured that the global `host` variable of the 3D JavaScript API, normally referencing the PDF document object, was `undefined`. In other examples, the `host` variable provided full access to the document object without any problems, and therefore the cause for this problem has initially been totally unclear.

A web enquiry showed that others met the same problem without knowing its reason either. Further investigations turned out that the `host` variable is solely undefined in those examples, where the generated PDF 3D annotation dictionary does not define the *NM* key, which represents its name. Since the *NM* key of annotation dictionaries was for the first time introduced in PDF 1.4, it has been declared as an optional entry. Seemingly, Adobe Acrobat requires this dictionary key, for 3D annotations, to establish a connection between the 3D context and the document object.

To prevent creators of FO3D based workflows from wondering about the undefined `host` variable, the FO3D plugin for Apache FOP always adds a name for 3D annotation objects. The annotation name is either defined in the corresponding attribute of the `fo3d:object-3d` element, or will be automatically generated by the FO3D plugin starting with "`__FO3DANNOT_`" and followed by an incrementing number. By the way, Adobe Acrobat does also automatically generate the names of 3D annotations.

6.4.4 PDF 3D Reference Dictionaries

As described in Section 4.3.1 for the `refgroup` attribute of the `object-3d` element, a 3D reference dictionary is used to enable multiple 3D annotations, sharing a single run-time instance of a specific 3D stream. Hence, it is theoretically possible to add multiple 3D annotations showing different views of the same 3D content and reflecting programmatic changes to the contained 3D objects.

Based on the PDF reference, the FO3D plugin successfully generates multiple 3D annotations that refer to a single 3D reference dictionary and the resulting PDF document shows the corresponding 3D content in all related viewports. This fact implies that the Acrobat PDF viewer application is capable of processing such 3D reference dictionaries, but additional tests showed that programmatic changes to one of these 3D annotations are not reflected by the others.

For example referring to the interactive visualization of technical 3D models, multiple views of the same object can tremendously enhance the perception of a viewer, which often is accociated with a better and even faster understanding. Hence, full support for 3D reference dictionaries would be highly desirable, but additional information on that topic is scarce, so this remains an open task for future releases of Adobe Acrobat or Adobe Reader.

Chapter 7

Concluding Remarks

The work presented in this thesis describes a new approach for integrating 3D content and related semantic metadata from arbitrary sources within a single PDF document. This is done by extending the traditionally two-dimensional XSL Formatting Objects (XSL-FO) standard with a new XML vocabulary, referred to as FO3D, to describe the integration of 3D objects and finally generate standard-compliant PDF documents with embedded 3D illustrations. The proposed workflow is exclusively based on open, approved and widely adopted XML technology and therefore the described approach provides a free alternative to expensive commercial applications, like for example, Adobe Acrobat.

Enriched 3D illustrations can be achieved by adding behaviours, which is in PDF done by JavaScript code, and appending of additional resources such as images or other 3D objects that can dynamically be loaded and used in the original scene. Starting from an XML-encoded description of a 3D model, multiple XSLT documents can be used to build completely different 3D visualizations including on-the-fly generated JavaScript code, additional resources and predefined views showing specific parts of the 3D content in more detail. This enables the successful fusion of plain 3D content and related information, for example, obtained from a database.

In order to show the practical application of the proposed extensions for 3D support in XSL-FO documents, a prototype implementation applied to the open-source FO processor Apache FOP has been developed. It works sufficiently well in practice and can easily be added to existing Apache FOP based XSL-FO workflows by just adding a single Java archive file to the classpath. This upgrades XML tools that rely on Apache FOP as their default FO processor, for example, *Altova XMLSpy* [9] and the *<oXygen/> XML editor* [44] to generate 3D-contained PDF documents too.

The described FO3D approach does not include the generation of 3D content itself and therefore it is mandatory that the used 3D objects are already available in form of a format that is supported by PDF, currently U3D (*Universal 3D*) and PRC (*Product Representation Compact*).

Since this work describes an extension to both theoretical and practical applications, it is heavily dependant on the underlying and used technology, such as the XML-related specifications, the corresponding PDF 1.7 reference, the XSL-FO processor and finally the PDF viewing application. Therefore, the limitations of this approach are also mainly accociated with the underlying technology. For example, Apache FOP 0.95 does not support the entire XSL-FO 1.1 W3C Recommendation and desirable features like `fo:float`, representing floating areas, are not yet available. On the client-side, the security concept of Adobe Reader does not permit network access for JavaScript code by default, which makes is nearly impossible to do things like:

- saving back a manipulated 3D scene to a database on a server,
- playing chess interactively in 3D, with another person or
- discussing 3D visualizations collaboratively for research purposes.

To enable such scenarios in the wide-spread Adobe Reader application, PDF documents have to be granted special rights, which can by done be the commercial software Adobe LiveCycle [5].

It is believed that the proposed XML-based approach is suitable for a great variety of documentation workfows, opening the door for a wide-range of visualizations in almost every field of application. This includes the area of automated PDF generation tools, for example, in web server and webservice environments, as well as individually created and 3D-supported scientific illustrations.

Since FO3D is neither based on a predefined input format, nor on a predefined representation of the 3D content in the resulting PDF document, it is offered as an enabling technology with a very high degree of freedom. Therefore, it can be utilized by users of different knowledge and education, from using the FO3D elements to add simple 3D objects to a more advanced developer writing his own PDF JavaScript applications and FO3D extensions.

Chapter 8

Outlook

Since FO3D does not provide support for all 3D related features that are defined in PDF 1.7 (ISO-32000-1), future releases may add additional functionality, such as the definition of *Markup Annotations* for predefined views of 3D artworks.

Since PDF 1.7 has been ISO standardized in 2008, the ISO is responsible for further developments of this document management standard. Currently, the PDF 2.0 (ISO/NP 32000-2) and PDF/A-2 (ISO/DIS 19005-2) standards are under development. The PDF/A-2 specification will be based on PDF 1.7 including 3D support, which enables PDF to be used for long-time archiving of 3D content too.

Referring to Adobe, their extensions to the PDF 1.7 standard, Extension Level 3 (Acrobat 9.0) and Extension Level 5 (Acrobat 9.1) have been submitted and accepted by the ISO for part two of the ISO 32000 specification. Concerning the 3D support in PDF documents, Adobe Extension Level 3 and Adobe Extension Level 5 add the following interesting new features:

- Support for the PRC file format, which can already be used with FO3D, but at least requirers Acrobat/Reader 9.0 to view the resulting PDF documents.
- *3D comments* can be defined and directly added as a textual annotation in 3D space. These comments are shown in the same list as conventional PDF comments and can also be found by the integrated search function.
- *3D Measurement/Markup Dictionaries* enable the visualization of distances in 3D space. Since this is an extension to the previously mentioned markup annotations, such measurement information is again linked with a specific predefined view of the 3D content.
- With *Rich Media Annotations*, support for SWF (*Shockwave Flash*) files is introduced and the functionality of embedded video, audio or other multimedia objects is expanded. It improves upon the existing 3D annotation structure to support multimedia file assets,

which for example enable a Flash video to be used as the texture of a 3D object. This does not only include the visual appearance but also interactive features of SWF files.

All of the previously described "new" features might also make sense in the FO3D PDF generation process and will possibly be integrated in future releases.

Altough, XML is praised to be human-readable, a person that is not familiar with XML and XSL-FO or with the proposed FO3D vocabulary, will possibly be overstrained by this approach. Therefore, easy to use authoring tools for working with FO3D are highly desirable to promote this approach, also for the use by non-technicians. Tools for the conversion of XML documents, including automated XSLT code generation, are already available, but additional support for sufficient high-level abstraction of behaviours in 3D space and a corresponding automated JavaScript code generation tool would be convenient for higher user adoption.

Since this approach and the described practical implementation is based on open standards and open-source software, this contribution to the field of publishing in 3D, will also be released under the open-source Apache License, Version 2.0, with the main intention to further encourage related developments. This work will soon be available at:

`http://fo3d.sourceforge.net`

In addition to further developments concerning the FO3D vocabulary or the FO3D Apache FOP plugin, the following enummeration outlines several application examples that could be realized on the basis of 3D-contained PDF documents in general and the work presented in this thesis in particular:

Education: On May 6, 2009, A. Schwarzenegger, governor of California, launched an initiative to make California the first nation to provide schools with free, open-source, digital textbooks for high school students [42]. 3D visualizations contained in PDF documents could be useful for subjects like maths, physics, chemistry, biology, geometry and also history to provide an in-depth understanding of mathematical functions, molecules, historical artefacts and so on. For exampe, a molecule could be dynamically visualized by just using two primitives, a unit ball and a unit cylinder that are attached to any given 3D scene as additional resources. By using appropriate JavaScript code, multiple colored and scaled spheres or cylinders can be arranged, according to a given chemical notation.

Construction Manuals: One of the most intuitive and useful fields of application is the generation of step-by-step construction manuals. These interactive instruction guides could, for example, be used for building up wardrobes or even small plastic LEGO games for children. In addition to the step-by-step explanation, predefined views can present particular procedures or parts in greater detail. Again, additional resources and JavaScript code can dynamically load and animate objects, such as screws or nails.

3D Mindmaps: Two-dimensional tag clouds are nowadays very common on web pages to provide a weighted collection of often used terms and therefore give a quick content overview. These terms could also be visualized in 3D space, by dynamically generating 3D-contained PDF documents on demand. Apart from just visualizing word clouds, complete three-dimensional mindmaps and also PDF documents having a 3D-table of contents could be realized.

Dynamic Fair Guides: Considering large trade fairs, enormous amounts of brochures and site maps are very expensive in print and possibly still inconvenient. Therefore, a web-based service could provide the ability to individually select interesting spots and events. Afterwards, the system calculates the best route according to the very specific interests of the current user. Finally, the user can download a dynamically generated PDF document that contains the suggested route, detailed information on the selected spots and a 3D model of the whole fairground, containing marks bidirectionally interlinked with the information about the current spot.

Remote 3D Surface Analysis: The dynamic generation of 3D PDF documents could also be used for providing a webservice where a user can upload a 3D model and will receive a PDF report that contains the given model, embedded as a 3D artwork, and additional evaluation results. Such a webservice might look for holes in 3D surfaces and provide a textual listing of all occurrences, each linked with a dynamically generated view that shows exactly where it can be found in the given 3D model. Such a system might be useful to ensure high quality of 3D models that shall be stored in a database repository.

PDF 3D Office Games: Last but not least, 3D PDF documents can also be used for interactive games having the major advantage that conforming PDF viewer applications are available on almost every computer. Therefore, such games could also be played in restricted setups like office PCs or computers in Internet cafes. In contrast to various JavaScript online games that could just be played in a two-dimensional web browser window, PDF documents could be used to enable portable 3D games.

Bibliography

[1] Adobe Systems Incorporated. *Extensible Metadata Platform (XMP)*. 2001. `http://www.adobe.com/products/xmp/`. [Online; accessed 02-April-2010]. (Cited on page 24.)

[2] Adobe Systems Incorporated. *JavaScript for Acrobat 3D - Adobe Acrobat SDK (Version 8.1)*. April 2007. `http://www.adobe.com/devnet/acrobat/javascript3d.html`. [Online; accessed 07-April-2010]. (Cited on pages 14, 47 and 67.)

[3] Adobe Systems Incorporated. *JavaScript for Acrobat API Reference*. April 2007. `http://www.adobe.com/devnet/acrobat/javascript.html`. [Online; accessed 06-April-2010]. (Cited on pages 13 and 48.)

[4] Adobe Systems Incorporated. *ISO 32000-1:2008 - Portable document format – Part 1: PDF 1.7*, 2008. (Cited on pages iv, 1, 8, 10, 35, 41, 57 and 69.)

[5] Adobe Systems Incorporated. *Adobe LiveCycle ES2*. 2010. `http://www.adobe.com/products/livecycle/`. [Online; accessed 05-Mai-2010]. (Cited on page 81.)

[6] Adobe Systems Incorporated. *Adobe Portable Document Format*. 2010. `http://www.adobe.com/pdf/`. [Online; accessed 23-March-2010]. (Cited on pages 1 and 9.)

[7] Adobe Systems Incorporated. *PDF Reference and Adobe Extensions to the PDF Specification*. 2010. `http://www.adobe.com/devnet/pdf/pdf_reference.html`. [Online; accessed 29-April-2010]. (Cited on page 8.)

[8] Adobe Systems Incorporated. *PRC File Format Specification - Version 8137*. 2010. `http://livedocs.adobe.com/acrobat_sdk/9/Acrobat9_HTMLHelp/API_References/PRCReference/PRC_Format_Specification/index.html`. [Online; accessed 06-April-2010]. (Cited on pages 2 and 16.)

[9] Altova GmbH. *XMLSpy - XML Editor for Modeling, Editing, Transforming, & Debugging XML Technologies*. 2010. `http://www.altova.com/xmlspy.html`. [Online; accessed 04-Mai-2010]. (Cited on page 80.)

[10] Keith Andrews. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria, July 2006. `http://ftp.iicm.edu/pub/keith/thesis/`. (Cited on page viii.)

[11] ANSI/ISO. *ANSI/NISO Z39.85 - The Dublin Core Metadata Element Set*, May 2007. `http://www.niso.org/standards/z39-85-2007/`. (Cited on page 23.)

[12] David G. Barnes and Christopher J. Fluke. *Incorporating interactive 3-dimensional graphics in astronomy research papers.* September 2007. doi:10.1016/j.newast.2008.03.008. (Cited on pages 2, 26 and 27.)

[13] Anders Berglund. *W3C Extensible Stylesheet Language (XSL) Version 1.1.* December 2006. `http://www.w3.org/TR/xsl11/`. [Online; accessed 22-March-2010]. (Cited on pages 2, 20, 21 and 34.)

[14] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. *W3C XML Path Language (XPath) 2.0.* January 2007. `http://www.w3.org/TR/2007/REC-xpath20-20070123/`. [Online; accessed 22-March-2010]. (Cited on page 18.)

[15] Tim Berners-Lee, Roy Fielding, and Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 2396).* Technical report, Network Working Group, January 2005. `http://www.ietf.org/rfc/rfc3986.txt`. (Cited on page 17.)

[16] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, and Henry S. Thompson. *W3C Namespaces in XML 1.0 (Third Edition).* December 2009. `http://www.w3.org/TR/2009/REC-xml-names-20091208/`. [Online; accessed 20-March-2010]. (Cited on page 17.)

[17] Tim Bray, Jean Paoli, Michael Sperberg-McQueen, Eve Maler, and François Yergeau. *W3C Extensible Markup Language (XML) 1.0 (Fifth Edition).* November 2008. `http://www.w3.org/TR/2008/REC-xml-20081126/`. [Online; accessed 20-March-2010]. (Cited on pages 2 and 16.)

[18] Nick Crofts, Martin Doerr, Tony Gill, Stephen Stead, and Matthew Stiff. *Definition of the CIDOC Conceptual Reference Model (version 5.0.2)*, 2005. Also ISO/PRF 21127:2006, available from http://cidoc.ics.forth.gr. (Cited on page 24.)

[19] Dublin Core Metadata Initiative. *Dublin Core.* 2010. `http://dublincore.org`. [Online; accessed 31-March-2010]. (Cited on page 23.)

[20] Ecma International. *Standard ECMA-363, Universal 3D File Format (4th Edition).* June 2007. `http://www.ecma-international.org/publications/standards/Ecma-363.htm`. [Online; accessed 20-March-2010]. (Cited on pages 2, 15 and 44.)

[21] Ecma International. *Universal 3D File Format (TC 43 Public Presentation).* April 2007. `http://www.ecma-international.org/activities/Communications/U3D-presentation-public.pdf`. [Online; accessed 09-March-2010]. (Cited on page 15.)

[22] Ecma International. *Standard ECMA-262, ECMAScript Language Specification (5th Edition).* December 2009. `http://www.ecma-international.org/publications/standards/Ecma-262.htm`. [Online; accessed 06-April-2010]. (Cited on page 12.)

[23] Ecrion Software Inc. *XF Rendering Server.* 2009. `http://www.ecrion.com/Products/XFRenderingServer/ProductInformation.aspx`. [Online; accessed 07-March-2010]. (Cited on page 28.)

[24] Steven J. Edwards. *Portable Game Notation Specification and Implementation Guide.* March 1994. `http://www.chessclub.com/help/PGN-spec`. [Online; accessed 24-March-2010]. (Cited on pages 51 and 73.)

[25] Alexander Grahn. *The movie15 LaTeX Package.* 2009. `http://www.ctan.org/tex-archive/macros/latex/contrib/movie15/`. [Online; accessed 20-April-2010]. (Cited on pages 1 and 28.)

[26] ISO. *ISO 8879:1986 - Standard Generalized Markup Language (SGML)*, 1986. (Cited on page 16.)

[27] ISO. *ISO/IEC 10179:1996 - Document Style Semantics and Specification Language (DSSSL)*, 1996. (Cited on page 18.)

[28] ISO. *ISO 19005 - Electronic document file format for long-term preservation*, October 2005. (Cited on page 9.)

[29] ISO. *ISO 15929 - Guidelines and principles for the development of PDF/X standards*, March 2008. (Cited on page 8.)

[30] ISO. *ISO 15930 - Prepress digital data exchange using PDF*, May 2008. (Cited on page 8.)

[31] ISO. *ISO 15836 - The Dublin Core metadata element set*, 2009. (Cited on page 23.)

[32] ISO. *ISO 15930 - Prepress digital data exchange using PDF*, December 2009. (Cited on page 8.)

[33] Ferraiolo Jon, Fujisawa Jun, and Dean Jackson. *W3C Scalable Vector Graphics (SVG) Version 1.1.* January 2003. `http://www.w3.org/TR/2003/REC-SVG11-20030114/`. [Online; accessed 12-April-2010]. (Cited on pages 3 and 34.)

[34] Rieko Kadobayashi. *Automatic 3D Blogging to Support the Collaborative Experience of 3D Digital Archives. Proceedings of the 8th International Conference on Asian Digital Libraries, ICADL 2005*, 1:109–118, 2005. doi:10.1007/11599517_13. (Cited on pages 2 and 27.)

[35] Michael Kay. *W3C XSL Transformations (XSLT) Version 2.0.* January 2007. `http://www.w3.org/TR/xslt20/`. [Online; accessed 20-March-2010]. (Cited on page 19.)

[36] Michael Kay. *XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer).* Wrox Press Ltd., Birmingham, UK, UK, 2008. ISBN 0470192747, 9780470192740. (Cited on pages 18 and 19.)

[37] Michael Kay. *SAXON - The XSLT and XQuery Processor.* 2010. `http://saxon.sourceforge.net`. [Online; accessed 29-April-2010]. (Cited on page 31.)

[38] Donald E. Knuth. *The TeXbook*. Addison-Wesley Professional, 1986. ISBN 0201134470. (Cited on page 21.)

[39] Pravin Kumar, Alexander Ziegler, Julian Ziegler, Barbara Uchanska-Ziegler, and Andreas Ziegler. *Grasping molecular structures through publication-integrated 3D models*. *Trends in Biochemical Sciences*, In Press, Corrected Proof. doi:10.1016/j.tibs.2008.06.004. `http://dx.doi.org/10.1016/j.tibs.2008.06.004`. (Cited on page 26.)

[40] John A. Kunze and Thomas Baker. *The Dublin Core Metadata Element Set*. Technical report, Network Working Group, August 2007. `http://www.ietf.org/rfc/rfc5013.txt`. (Cited on page 23.)

[41] Laurens Leurs. *The history of PDF*. 2010. `http://www.prepressure.com/pdf/basics/history`. [Online; accessed 29-April-2010]. (Cited on page 6.)

[42] State of California. *Gov. Schwarzenegger Launches First-in-Nation Initiative to Develop Free Digital Textbooks for High School Students (GAAS:223:09)*. May 2009. `http://www.gov.ca.gov/press-release/12225/`. (Cited on page 83.)

[43] Martin Strobl, René Berndt, Sven Havemann, and Dieter W. Fellner. *Publishing 3D content as PDF in cultural heritage*. 2009. (Cited on pages 2, 27, 65, 68 and 69.)

[44] SyncRO Soft Limited. *<oXygen/> XML Editor*. 2010. `http://www.oxygenxml.com`. [Online; accessed 04-Mai-2010]. (Cited on page 80.)

[45] The Apache Software Foundation. *Apache FOP (Formatting Objects Processor)*. 2010. `http://xmlgraphics.apache.org/fop/`. [Online; accessed 23-March-2010]. (Cited on pages 3, 29, 35 and 52.)

[46] The Apache Software Foundation. *Xalan-Java*. 2010. `http://xml.apache.org/xalan-j/`. [Online; accessed 29-April-2010]. (Cited on page 31.)

[47] The Text Encoding Initiative. *TEI Guidelines*. 2010. `http://www.tei-c.org/Guidelines/`. [Online; accessed 07-Mai-2010]. (Cited on page 24.)

[48] Visual Computing Lab. *MeshLab*. 2010. `http://meshlab.sourceforge.net`. [Online; accessed 22-April-2010]. (Cited on page 69.)

[49] Visual Technology Services Ltd. *PDF3D*. 2010. `http://www.pdf3d.co.uk`. [Online; accessed 28-April-2010]. (Cited on page 28.)

[50] W3C. *Mathematical Markup Language (MathML^{TM}) Version 1.01*. July 1999. `http://www.w3.org/1999/07/REC-MathML-19990707`. [Online; accessed 12-April-2010]. (Cited on page 34.)

[51] W3C. *Resource Description Framework (RDF)*. February 2004. `http://www.w3.org/RDF/`. [Online; accessed 22-March-2010]. (Cited on page 24.)

[52] W3C. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. September 2009. `http://www.w3.org/TR/CSS2/`. [Online; accessed 20-March-2010]. (Cited on page 20.)

[53] John Warnock. *The Camelot Project*. 1991. `http://www.planetpdf.com/planetpdf/pdfs/warnock_camelot.pdf`. [Online; accessed 22-March-2010]. (Cited on page 5.)

Glossary

ANSI	American National Standards Institute
API	Application Programming Interface
CAD	Computer Aided Design
CRM	Conceptual Reference Model
DC	Dublin Core
DCMI	Dublin Core Metadata Initiative
DSSSL	Document Style Semantics and Specification Language
ECMA	European Computer Manufacturers Association
DAML+OIL	Darpa Agent Markup Language + Ontology Inference Layer
FO	Formatting Objects
FOP	Formatting Objects Processor
FO3D	3D Formatting Objects
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
ICOM	International Council of Museums
ICOM-CIDOC	ICOM - International Committee for Documentation
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
JPEG	Joint Photographic Experts Group (ISO/IEC-10918-1)
KIF	Knowledge Interchange Format
MathML	Mathematical Markup Language
MIF	Maker Interchange Format
NISO	National Information Standards Organization
OPI	Open Prepress Interface
OWL	Web Ontology Language
PRC	Product Representation Compact
PDF	Portable Document Format
PDF/A	PDF/Archive
PDF/E	PDF/Engineering
PDF/X	PDF/Exchange

PGN	Portable Game Notation
PNG	Portable Network Graphics
RDF	Resource Description Framework
RFC	Request for Comments
RTF	Rich Text Format
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
STEP	STandard for the Exchange of Product model data
SVG	Scalable Vector Graphics
SWF	Shockwave Flash
TC	ISO Technical Committee
TEI	Text Encoding Initiative
TIFF	Tagged Image File Format
U3D	Universal 3D
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	XML Stylesheet Language
XSLT	XSL - Transformation
XSL-FO	XSL - Formatting Objects
XMP	Extensible Metadata Platform