Berufsakademie Sachsen
Staatliche Studienakademie Leipzig

Dynamische Darstellung von Text und Grafik im Video

Studienarbeit
in der Studienrichtung Informatik

Eingereicht von: Christian Gräfe
Hahnemannstraße 22
04177 Leipzig
Seminargruppe: IT2005
Matrikelnummer: 205830

Leipzig, den 28.02.08

Inhaltsverzeichnis

1 Einführung und Problemstellung

Um Videodateien in verschiedenen Formaten auf einem Windows-PC wiederzugeben braucht es nicht viel. Seit es DirectShow gibt sind die Möglichkeiten nahezu unbegrenzt. Wenn der richtige Codec installiert ist, läuft alles fast von alleine. Man öffnet die Videodatei und im Hintergrund sorgt DirectShow dafür, das der Benutzer auf seinem Bildschirm das Video sieht und sich da drin bewegen kann.

Was aber ist, wenn das nicht mehr ausreicht? Wenn der Benutzer mehr sehen will als nur das Video. Wie zum Beispiel einen Untertitel für Hörgeschädigte oder Hintergrundinformationen zur laufenden Szene.

Im Bereich Sport werden Videoanalysen immer wichtiger, hier können frühzeitig Fehler im Training erkannt und Spieltaktiken ausgewertet werden. Durch das Video und die Anmerkungen des Trainers dazu, können Sportler ihre Leistung ständig verbessern. Bei den Anmerkungen des Trainers wird es sich meist um Linien, Pfeile, Kreise und andere einfache geometrische Formen handeln, welche durch einen kurzen Text erläutert werden. Um den Lernprozess des Sportler zu unterstützen, ist es von Vorteil, diese Anmerkungen direkt im laufenden Video darzustellen. So kann der Sportler sich das Video später Szene für Szene nochmal ansehen und die Anmerkungen verinnerlichen.

Um die oben genannten Beispiele zu ermöglichen, gibt es im Grunde zwei Ansätze. Eine Möglichkeit wäre es, die erweiterten Informationen direkt in der Videodatei zu speichern. So kann der Untertitel statisch in das Videobild integriert sein bzw. je nach verwendetem Videocodec und Containerformat in einer extra Datenspur. Eine zweite und bessere Variante ist es, die erweiterten Informationen in einer extra Datei, unabhängig vom Video zu speichern. So können die Informationen nicht nur schnell erweitert und angepasst werden, es ist auch möglich, die Informationen unabhängig von dem verwendeten Videocodec zur Verfügung zu stellen.

Die Ermöglichung eines solchen Verfahrens, zur dynamischen, videocodecunabhängigen Darstellung von Text und Grafik im Video, ist Ziel dieser Arbeit. Im Verlauf dieser Arbeit werden Grundlagen erklärt, Teilprobleme erläutert und Lösungsansätze vorgestellt. Im letzten Drittel wird eine mögliche Umsetzung, mit dem Ziel eines Prototypen, dargestellt.

2 Grundlagen

2.1 DirectX

Als DirectX wird eine Reihe von COM-Schnittstellen bezeichnet, die von Microsoft zur Verfügung gestellt werden. Es handelt sich dabei um API's für verschiedene multimediale Zwecke, wie die Grafikdarstellung oder Soundwiedergabe. Die aktuelle Version ist 10.1 (unter XP allerdings nur 9.0c).

Für die Grafikdarstellung in 2D und 3D bietet *DirectX Graphics* eine sehr Umfangreiche API, welche nicht nur für Computer- bzw. Konsolenspiele genutzt wird, sondern auch für CAD- und andere grafisch aufwendige Anwendungen. Der direkte Zugriff auf die Grafikhardware sorgt für eine möglichst hohe Effektivität der Routinen. Werden Funktionen nicht durch die vorhandene Hardware unterstützt, werden diese anderweitig durch DirectX emuliert.

DirectSound ist verantwortlich für die Wiedergabe und das Mischen von Soundeffekten. Es passt sich automatisch dem Leistungsspektrum der installierten Soundkarte an.

Mit Hilfe von *DirectMusic* können MIDI-Dateien wiedergegeben werden. Diese werden, unabhängig von der Hardware, durch einen Software-Synthesizer auf DLS2 Basis verarbeitet.

Für die Abfrage und die Kommunikation mit Eingabegeräten stellt *DirectInput* die benötigten Routinen zur Verfügung. DirectInput umgeht, wie die anderen DirectX Schnittstellen auch, das Windows Message System und greift direkt auf die angeschlossenen Hardware zu.

Als Unterstützung für die Netzwerkkommunikation ist *DirectPlay* gedacht. Da es aber gegenüber Windows Sockets einen größeren Overhead produziert wird es unter Spieleentwicklern eher selten verwendet.

DirectShow ist seit kurzem nicht mehr offizieller Bestandteil des DirectX SDK. Es ist für die Verarbeitung von Audio- und Videodaten konzipiert und ist jetzt in das Windows-Plattform SDK eingegangen. DirectShow ist ein wesentlicher Bestandteil dieser Arbeit und wird im folgenden Kapitel näher erklärt.[1]

1 Wikipedia: DirectX [1]

2.2 DirectShow

2.2.1 Allgemeines und Geschichtliches

Die meisten Abspielprogramme für Audio oder Video unter Windows, wie zum Beispiel der MediaPlayer, nutzen DirectShow. Es stellt, ähnlich wie QuickTime auf dem Mac oder gstreamer unter Linux, Funktionen für das Lesen, Verarbeiten und Schreiben von Audio- und Videodaten zur Verfügung. Es bietet außerdem Decoder für verschiedene Standardformate, wie zum Beispiel AVI, ASF, MPEG, WMV sowie MP3, WMA und WAV.[2]

DirectShow gibt es schon länger als DirectX. Microsoft veröffentlichte ende 1992 *Video for Windows*. Es legte den Grundstein für die Wiedergabe von Videodaten auf einem Windows-Betriebssystem. Video for Windows wurde 1996 durch das COM-Basierte ActivMovie ersetzt, welches schließlich 1997 in DirectX5 einging. Nach der Umbenennung in DirectShow wurde es erst als Hauptbestandteil des DirectMedia SDK verbreitet und mit der Veröffentlichung von DirectX7 schließlich vollständig integriert. Seit 2005 ist DirectShow nicht mehr Bestandteil des DirectX SDKs. Es ist nun im Windows Plattform SDK enthalten und soll voraussichtlich durch *Media Foundation* abgelöst werden.[3]

2.2.2 Filterkonzept

Da DirectShow, wie die anderen DirectX Schnittstellen auch, auf dem *Component Object Model* (COM) basiert, ist es sehr leicht zu erweitern. In DirectShow werden so genannte *Filter* für die Wiedergabe der Daten genutzt. Bei jedem Filter handelt sich sich um ein COM-Object. Sie werden zur Verarbeitung der Daten in einem *Filtergraphen* kombiniert. Es gibt mehrere Filtertypen[4]:

- *Source Filter* erzeugen den Inputstream für den Filtergraphen. Die Daten können aus einer Datei, einem Internet- bzw. Netzwerkstream oder von einem angeschlossenem Gerät kommen. Durch die enge Zusammenarbeit mit dem Windows Driver Model (WDM) kann multimediale Peripherie direkt als Source Filter ausgewählt werden. Source Filter können nochmal in zwei Typen unterteilt werden[5]:

2 MSDN: Introduction to DirectShow [2]
3 Wikipedia: DirectShow [3]
4 nach [4], Part I, Chapter 1, Abs.: Filters
5 nach [4] Part III, Chapter 12, Abs.: Source Filter Types

    - *Push*: es werden fortlaufend Daten auf den Output Pin ausgegeben. Dies trifft für Live-Quellen, wie ein Internetstream, eine Webcam oder eine Kamera zu.
    - *Pull*: die Daten werden bei Bedarf an den Output Pin gelegt. Dies trifft für Datei-Quellen zu, in ihnen kann sich vor und zurück bewegt werden.
- *Transform Filter* verändern den Datenstrom, der durch sie hindurchgeht. Hier geschieht die eigentliche Arbeit von DirektShow, der Filter bekommt Daten aus einem anderen Filter, verändert diese und gibt sie an den nächsten Filter weiter. Transfom Filter werden zum Decodieren und Encodieren eingesetzt. Unter die Gruppe der Transform Filter fallen noch zwei weitere Filtertypen:[6]
    - Der *Splitter Filter* teilt einen Eingangsstrom in mehre Ausgangsströme auf. Meist handelt sich dabei um die Zerlegung in Audio und Video.
    - Das Gegenstück dazu ist der *Mux Filter* er verbindet mehrere Eingangsströme zu einem Ausgangssignal. Es können Audio- und Videodaten verbunden oder mehrere Videosignale kombiniert werden.
- *Renderer Filter* geben den Datenstrom aus. Die Daten werden auf den Bildschirm, den Lautsprecher oder sogar in eine Datei ausgegeben. Da DirectShow auch DirectDraw (DirectX Graphics) und DirectSound nutzt, kann es die Daten ohne Umwege direkt auf die Hardware ausgeben.

Ein Filtergraph besteht immer aus mindestens einem Source Filter und einem Renderer Filter, meist kommen noch ein oder mehr Transform Filter zum Einsatz.

Die Filter sind verbunden und kommunizieren über *Pins*, welche vom Filter genau definiert sind. Source Filter haben nur Output Pins und Render Filter nur Input Pins. Transform Filter haben mindestens einen Input und einen Output Pin. Dabei kann nicht jeder Output Pin mit irgend einem Input Pin verbunden werden. Es wird vom Filter genau festgelegt, welche Art von Daten über den Pin ausgetauscht wird und auf welche Art und Weise der Austausch geschieht. Dadurch ist es zum Beispiel nicht möglich einen Output Pin der MPEG-1 Daten liefert mit einem Input Pin zu verbinden, der Digital Video (DV) Daten voraussetzt. Solche eine Verbindung würde den Datenstrom unbrauchbar machen.[7]

6 nach MSDN: About DirectShow Filters [5]
7 nach [4] Part I, Chapter 1, Abs.: Connections Between Filters

2.2.3 Intelligente Verbindung von Filtern

Um die Arbeit mit den Filter zu vereinfachen, stellt DirectShow *Intelligente Verbindung* zur Verfügung. Man übergibt dem Filtergraphen einen Source Filter. Der Output Pin dieses Filters kann automatisch so aufgelöst werden, das am Ende ein vollständiger Graph entsteht, welcher die Daten ausgibt. Der Filtergraph sucht nach einem passendem Filter, mit dem verbunden werden kann, und fügt diesen in den Graphen ein. Wenn man dem Filtergraphen zum Beispiel einen Source Filter übergibt, welcher eine AVI-Datei liest, sieht der Filtergraph wie in Abbildung 1 aus, vorausgesetzt die AVI-Datei enthält DV-Daten.

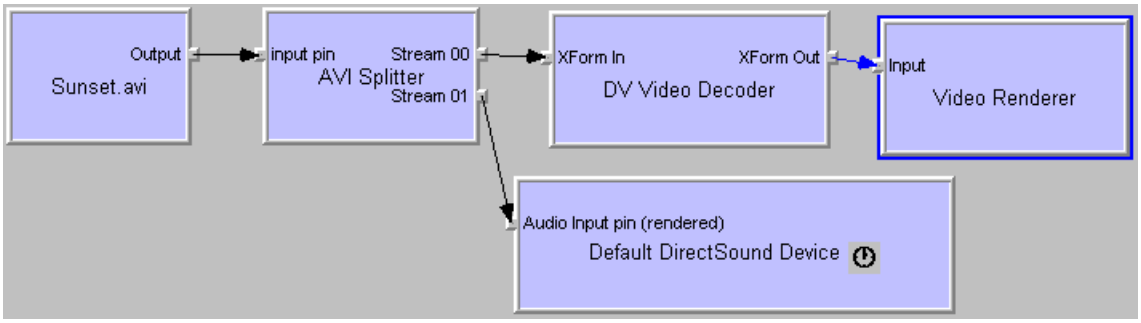


Abbildung 1: Filtergraph einer AVI Datei mit DV-Daten

2.2.4 Zeit im Filtergraphen

Der Filtergraph ist dafür verantwortlich, das die Filter synchron ablaufen. Um dies zu erreichen ist eine *Referenzzeit* notwendig. In Abbildung 1 wird diese Referenzzeit vom Filter "Default DirectSound Device" zur Verfügung gestellt, deshalb ist dieser mit einer kleinen Uhr gekennzeichnet. Enthält der Filtergraph einen Push Source Filter stellt immer dieser die Referenzzeit. Bei der Referenzzeit handelt es sich immer um 100 Nanosekunden Intervalle.[8]

8 nach MSDN: Reference Clocks [6]

2.3 XML

2.3.1 Allgemein

Die Extensible Markup Language (XML) ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien. XML ist eine vereinfachte Teilmenge von SGML.

Die XML-Spezifikation wurde erstmals 1998 vom World Wide Web Consortium (W3C) herausgegebene und mehrfach überarbeitet. Die Spezifikation definiert eine Metasprache, auf deren Basis weitere Sprachen aufbauen. Diese Sprachen werden durch inhaltliche und strukturelle Einschränkungen in Form von DTDs oder XML-Schemas definiert. Beispiele für solche Sprachen sind XHTML, SOAP, RSS oder die XML-Schema (XSD) selber.[9]

Im Zuge dieser Arbeit sind zwei dieser Sprachen von besonderer Bedeutung, daher werden sie in den folgenden Kapiteln näher erläutert. Es handelt sich dabei um die Transformationssprache XSLT und die Vektorgrafiksprache SVG.

2.3.2 XSLT

XSL Transformation (XSLT) ist eine Programmiersprache zur Transformation von XML-Dokumenten. XSLT baut auf der logischen Baumstruktur eines XML-Dokumentes auf und dient zur Definition von Umwandlungsregeln. XSLT-Programme, so genannte XSLT-Stylesheets, sind dabei selbst XML-Dokumente.[10]

Die Stylesheets werden von spezieller Software, den XSLT-Prozessoren, eingelesen und auf ein oder mehrere XML-Dokumente angewandt. Durch die Verarbeitung entsteht eine neues Dokument. Bei der Ausgabe handelt es sich meist ebenfalls um ein XML-Dokument, es kann allerdings auch eine Textdatei oder sogar eine Binärdatei sein.

Das wichtigste Element in XSLT ist die *Template Rule* (Schablonen Regel). Mit Hilfe dieser Regel werden die Tags aus dem Eingangs-XML verarbeitet. Darüber hinaus gibt es Konstrukte für die bedingte Ausführung und die Sortierung. Variablen sind ebenfalls möglich, jedoch unter der Einschränkung, dass sie nur definiert und nicht verändert werden können! Im Anhang (Seite I) findet sich ein Beispiel, für ein einfaches XSL-Stylesheet.

9 nach Wikipedia: Extensible Markup Language [7]
10 nach Wikipedia: XSL Transformation [8]

2.3.3 SVG

Scalable Vector Graphics (SVG, deutsch Skalierbare Vektorgrafiken) ist ein Standard zur Beschreibung zweidimensionaler Vektorgrafiken in der XML-Syntax. SVG wurde im September 2001 vom W3C als Empfehlung veröffentlicht.[11]

In SVG werden einfache geometrische Figuren durch XML-Tags beschrieben. Bei diesen Figuren handelt es sich um das Rechteck (<rect>), der Kreis (<circle>) bzw. die Ellipse (<ellipse>), so wie die Linie (<line>), die Mehrpunkt-Linie (<polyline>) und das Polygon (<polygon>). Das wichtigste und mächtigste Element ist der Pfad (<path>), aus ihm lassen sich alle Grundelemente aufbauen. Ebenso kann in SVG auch Text gespeichert werden (<text>).

Mithilfe von SMIL sind in SVG auch Animationen möglich. Wie bei den meisten XML-Dokumenten kann die DOM mittels ECMAScript manipuliert werden. Auf diese weiße ist eine Interaktion und Manipulation der dargestellten Objekten möglich.

Alle Elemente in SVG lassen sich über das Attribut "transform" beliebig verändern. Strecken, Spiegeln und Rotieren sind ebenso möglich wie auch eine komplexe Matrix-Transformation. Darüber hinaus gibt es auch Grafikfilter, wie sie aus anderen Bildbearbeitunsprogrammen bekannt sind.

11 Wikipedia: Scalable Vector Graphics [9]

2.3.4 XML-Schema (XSD)

XSD steht für XML Schema Definition, dabei handelt es sich um eine Beschreibungssprache für den strukturellen Aufbau von XML Dokumenten. Es bietet gegenüber DTD (Document Type Definition) den großen Vorteil, dass es selber ein XML Dokument ist. Darüber hinaus stellt es mehr Datentypen und andere Konstrukte zur Verfügung um die Struktur einer XML-Datei zu spezifizieren.[12]

XSD unterscheidet bei den Datentypen zwischen atomaren, einfachen Datentypen (simpleType) und zwischen zusammengesetzten Datentypen (complexType). Bei den einfachen Datentypen handelt es sich um die gleichen Typen, wie sie auch in Programmiersprachen vorkommen. Integer (xsd:integer), Float (xsd:float), Boolean (xsd:boolean) oder String (xsd:string) sind einige von ihnen.

In Ergänzung zu den einfachen Typen bietet der complexType die Möglichkeit, Wurzelelemente, Ketten von Kindelementen, sowie Attribute zusammenhängend zu definieren. Ebenso ist es möglich durch Ableitung einen neuen Datentyp zu definieren. Durch Einschränken (xsd:restriction) bzw. Erweitern (xsd:extension) des Basistypen wird der neue Typ definiert.

Durch Einschränkung eines simpleType ist es zum Beispiel möglich den Wertebereich für eine Zahl oder die Länge eines Strings zu begrenzen. Des Weiteren kann auch eine Auswahlliste von Werten vorgegeben werden. Darüber hinaus bietet XSD die Möglichkeit eindeutige Schlüssel, vergleichbar mit den Primärschlüsseln einer relationalen Datenbank, zu definieren.

12 nach Wikipedia: XML Schema [10]

3 Konzeptvorbereitung

3.1 Vorbemerkungen

In diesem Kapitel werden wichtige Punkte aufgeführt, die vor der eigentlichen Konzeptentwicklung betrachtet werden müssen. Abhängig von diesen Punkten kann ein genaueres Programmkonzept entwickelt werden. Schwerpunkt dieser Betrachtung bilden drei Aspekte. Zum einen wie die Daten geöffnet werden, wie sie verarbeitet werden und wie die verarbeiteten Daten schlussendlich im Videobild eingeblendet werden.

Als Grundvoraussetzung wird angenommen, das die zusätzlichen Daten in einer extra Datei, in einem XML-Format gespeichert werden. Die genaue Spezifikation dieses XML-Formates ergibt sich erst mit dem genauen Konzept. Die Datei mit den XML-Daten hat eine definierte Dateiendung, damit sie als solch eine "Datei mit Zusatzinformationen" erkannt werden kann.

3.2 Datei öffnen

Als Einstiegspunkt für das Programm ist das Öffnen der Datei von besonderer Bedeutung. Beim Öffnen der Datei muss der Filtergraph erzeugt werden, welcher die Daten verarbeitet und Wiedergibt. Damit das Programm weiß, welche Daten es wiedergeben soll, muss es wissen, wo es die nötigen Informationen findet. Hierbei sind zwei Varianten möglich. Zum einen kann das Video geöffnet werden und zum anderen die XML-Datei mit den Zusatzdaten.

Die Videodatei öffnen

Wenn das Programm die Videodatei öffnet, weiß es nicht, ob und wo es die Datei mit den Zusatzinformationen findet. Wenn man davon ausgeht, das sich die Datei mit den Zusatzdaten im gleichen Verzeichnis wie die Videodatei befindet, sind mehrere Verfahren möglich

Es wäre denkbar, dass das Programm alle XML-Dateien mit der entsprechenden Endung überprüft, um festzustellen ob eine Datei mit Zusatzinformationen existiert. So ein Verfahren kann das ganze Programm allerdings negativ beeinflussen. Wenn es sehr viele solcher Dateien gibt, wird das ganze sehr Zeitaufwendig. Wenn dann auch noch eine der Dateien zwar die passende Dateiendung besitzt, aber eigentlich keine XML-Datei ist, kann dies schnell zu einem Fehler, wenn nicht sogar zum Abbruch des Programms füh-

ren. Problematisch ist auch, wenn zwei passende Dateien gefunden werden. Wie entscheidet man dann welche verwendet werden soll? Wie man sieht überwiegen die Nachteile bei solch einem Verfahren. Es wird unmöglich einen erfolgreichen Programmablauf zu garantieren.

Die einfachste Lösung ist manchmal die Beste. Wenn man die XML-Datei genau so benennt, wie die Videodatei, umgeht man die meisten der oben genannten Probleme. Als Beispiel: die Videodatei heißt XYZ.avi, dann muss die Datei mit den Zusatzdaten XYZ.xxx benannt werden, wobei xxx für die noch undefinierte Dateiendung der XML-Datei steht.

Bei dieser Variante ergibt sich allerdings eine Einschränkung. Es ist nicht möglich zwei Videodateien mit dem gleichen Namen, aber unterschiedlichen Dateiendungen, mit Zusatzdaten zu versorgen. Es kann darüber hinaus passieren, dass die Zusatzinformationen angezeigt werden, obwohl sie für die andere Videodatei bestimmt sind.

Um dies zu umgehen gibt es zwei Möglichkeiten. Entweder man vermerkt die Dateiendung in den Zusatzdaten oder man erweitert die Benennung der Zusatzdatendatei um die Dateiendung der Videodatei. Um bei dem obigen Beispiel zu bleiben, die Datei mit den Zusatzdaten würde hier dann XYZ.avi.xxx heißen. Wenn man die Dateiendung nur in den Zusatzdaten vermerkt, besteht immer noch das Problem, dass es nur eine Datei mit Zusatzdaten geben kann. Hier ist abzuwägen, was einem besser gefällt, und ob man diese Einschränkung hinnehmen kann.

Die XML-Datei öffnen

Wenn man davon ausgeht, dass das Programm die Zusatzdaten nur anzeigt, wenn die entsprechende Datei übergeben wird, wird die Sache etwas einfacher. In den Zusatzdaten muss nur die Videodatei explizit angegeben sein.

Diese Variante ist zu bevorzugen, wenn sich die zwei Dateien nicht im gleichen Verzeichnis befinden. In der XML-Datei kann in diesem Fall der absolute bzw. relative Pfad zur Videodatei vermerkt werden.

Optimal wäre es natürlich, wenn beides geht, also die Videodatei oder die XML-Datei übergeben werden kann. Hier ist, wie bereits erwähnt die Variante mit den gleichen Dateinamen am besten geeignet. Die Dateien müssen sich dann aber zwingend im gleichen Verzeichnis befinden.

3.3 XML-Daten verarbeiten

3.3.1 Wie die Zusatzdaten gespeichert werden

Die Zusatzdaten werden in einer XML-Datei gespeichert. Hierbei ergeben sich zwei Aspekte, die von entscheidender Bedeutung für den ganzen Programmablauf sind. Zum einen muss geklärt werden, wie die Zusatzdaten den einzelnen Szenen im Video zugeordnet werden können, und zum anderen muss definiert werden, mit welcher Syntax die Zusatzdaten überhaupt gespeichert werden.

Ersteres ist Wichtig, damit die Daten gefiltert werden können. Es sollen ja nicht immer alle Daten angezeigt werden. Wenn zum Beispiel ab der fünften Sekunde, für drei Sekunden lang ein Rechteck angezeigt werden soll, muss dies irgendwie erfasst werden. Es muss also ein Weg gefunden werden, wie diese Zeitangaben im XML gespeichert und anhand dieser Angaben, die Informationen zur richtigen Zeit eingeblendet werden können. (siehe Kapitel 3.3.2)

Der zweite Aspekt ist noch wichtiger, da das Programm wissen muss, was es überhaupt anzeigen soll. Es muss bekannt sein, was ein Element in den XML-Daten bedeutet. Also das zum Beispiel ein <rect>-Tag ein Rechteck ist. Und es muss bekannt sein, was die Attribute eines Elementes angeben. Wenn also das <rect>-Tag ein Attribut "width" hat, dass dieses die Breite des Rechtecks angibt. Diese Angaben sind nötig, da die Zusatzdaten aus einem Text- bzw. Vektorformat in ein Pixelformat konvertiert oder besser gesagt gerendert werden müssen. (siehe Kapitel 3.3.3)

Aus diesen zwei Aspekten ergibt sich eine Vorgabe für das zu verwendende XML-Schema. Man muss bei diesen Aspekten genau abwägen, was das Programm können soll und wie erweiterbar es ist. Wenn erst einmal ein XML-Schema erstellt wurde, zieht eine Änderung dieses Schemas fast immer eine Änderung im Programm nach sich. In den folgenden zwei Kapiteln werden daher mehrere Mögliche vorgestellt, wie die Daten in XML-Form gespeichert werden können.

3.3.2 Die XML-Daten nach der Zeit Filtern

Wenn man die Zusatzinformationen für einen bestimmten Zeitpunkt im Video haben möchte, muss man die XML-Daten danach Filtern bzw. Durchsuchen. Um solch ein Filter zu beschleunigen, muss die Struktur der XML-Datei möglichst optimal ausgelegt sein. Hierfür ergeben sich zwei Möglichkeiten.

Wenn man die Zusatzinformationen für eine bestimmte Szene unter einem Knoten zusammenfasst. Also das man ein <szene>-Knoten hat und dieser hat als Kindknoten die Informationen die eingeblendet werden sollen. Der <szene>-Tag benötigt dann Attribute wann und wie lang diese Informationen angezeigt werden sollen. Dieses Verfahren kann allerdings im schlimmsten Fall dazu führen, dass für jede Information ein <szene>-Tag angegeben werden muss, da eine Zusammenfassung durch unterschiedliche Einblend- und Ausblendzeiten unmöglich ist.

Alternativ dazu setzt man bei jedem XML-Tag des Dokumentes Attribute für die Zeit. Die Informationen werden also nicht durch ein <szene>-Tag strukturiert, sondern befinden sich alle auf der gleichen Hierarchie-Ebene.

Hier muss also abgewogen werden, ob man viele vereinzelte Informationen darstellen will oder ob die Informationen meist gleichzeitig angezeigt werden. Durch die gute Strukturierung des XML ist in den meisten Fällen aber die Variante mit den <szene>-Tags zu bevorzugen.

Wichtig für die Filterung bzw. Suche sind die schon erwähnten Attribute mit den Zeitangaben. Um den Zeitpunkt zu markieren, wann die Information eingeblendet werden soll, ist zwingen eine absolute Zeitangabe im Video notwendig. Für das Ausblenden der Information kann eine Zeit, relativ zu der Anfangszeit, oder ebenfalls eine absolute Zeit verwendet werden. Um den Rechenaufwand im Programm allerdings so gering wie möglich zu halten, bietet es sich an hier ebenfalls die absolute Zeit zu verwenden.

Das verwendete Zeitformat dieser Attribute sollte sich an der Zeitangabe in DirectShow orientieren. Um ein meist nötiges Parsen der Zeitangaben zu vermeiden, sollte es sich dabei um eine Ganzzahl handeln, hier könnte zum Beispiel die Referenzzeit genutzt werden. Diese wird aber bei längeren Videos sehr groß, eine Sekunde wäre dann eine 7-stellige Zahl! Da man die Informationen nicht wirklich so übergenau darstellen kann und muss, sollte als Zeiteinheit Frames verwendet werden.

3.3.3 XML-Daten Rendern

Der wichtigste Punk, der beachtet werden muss, wenn man ein Schema für die XML-Datei erstellen will, ist das Rendern der Zusatzinformationen. Da die Informationen in Textform vorliegen, aber die Ausgabe grafisch erfolgt, muss ein Weg für die Umwandlung gefunden werden. Diese Umwandlung erfolgt über einen so genannten Renderer. Es gibt nun zwei Möglichkeiten so einen Renderer einzubinden.

Zum einen kann ein eigener Renderer eingebunden werden. Dieser ist dann genau auf die eigenen Bedürfnisse zugeschnitten. Hierbei hat meine freie Hand für die Entwicklung eines XML-Schemas für die Zusatzdaten. Die Routinen, die die XML-Daten in eine Grafik umsetzen, müssen hierbei allerdings selber geschrieben werden.

Zweite Möglichkeit ist ein schon vorhandener Renderer. Hier kann zum Beispiel ein Renderer für SVG Daten eingesetzt werden. Das Schema der XML-Datei orientiert sich dann an dem SVG-Standard. Hierbei entfällt das Schreiben eigener Routinen für die Umsetzung von Text nach Grafik.

Mit Hilfe des SVG-Standards wird ein großer Teil der benötigten Anzeigemöglichkeiten abgedeckt. Unter Verwendung von SVG hat man außerdem einen sehr großen Handlungsspielraum, bei der Gestaltung der dargestellten Inhalte. Farbverläufe und Alphablending werden ebenso wie einfache Textdarstellung von den meisten SVG-Renderern standardmäßig unterstützt. Darüber hinaus gibt es gute Grafikprogramm die SVG-Code erzeugen können.

Der Nachteil bei der Verwendung eines SVG-Renderers besteht allerdings in der Abhängigkeit des Programms von dieser Komponente. Darüber hinaus sind des weiteren auch lizenzrechtliche Aspekte zu beachten.

Wenn man sich entschließt einen eigenen Renderer zu verwenden, sollte man bei den Überlegungen für das XML-Schema den SVG-Standard mit einbeziehen. Wenn man das Schema für die XML-Daten an SVG anlehnt bzw. eine Teilmenge davon nutzt, kann man später das Programm bei Bedarf auf einen SVG-Renderer Umstellen. Des weiteren spart man sich viele der Überlegungen, was einzelnen Tags nun bedeuten könnten und was sie alles können.

3.4 Einblendung Daten in das laufende Video

Das Rendern der Zusatzdaten in ein Pixelformat sind Ausgangspunkt für das Einblenden dieser Informationen in das Video. Die Pixeldaten werden über das Videobild gelegt. Dieser Vorgang kann an zwei Stellen im Filtergraph geschehen. Entweder in einem Eigenen Filter oder direkt im VideoRenderer-Filter, wenn dieser die Möglichkeiten dafür bietet.

Verwendung eines Mux-Filter

Ein Mux-Filter bietet den Vorteil, dass man komplett unabhängig von dem verwendeten VideoRender-Filter ist und den Datenstrom in weiteren Filtern bearbeiten kann. In Abbildung 2 wird dies durch gestrichelte Linien dargestellt.

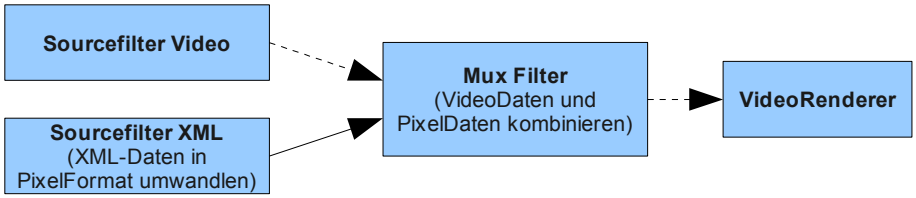


Abbildung 2: Mux-Filter im Filtergraphen

In DirectShow gibt es einen Overlay-Mixer-Filter. Dieser ist Primär für die Wiedergabe von DVD-Video gedacht. Er ermöglicht die Einblendung von Untertiteln. Darüber hinaus bietet er aber auch die Möglichkeit Grafikdaten über das Videobild zu legen.

Bei Verwendung dieses Filters ist eine möglichst hohe Kompatibilität, bezüglich des Betriebssystems gewährleistet.

Verwendung des VideoRenderer-Filters

Microsoft bietet mit den zwei Filtern VMR-7 und VMR-9 ebenfalls die Möglichkeit Grafikdaten über das Videobild zu legen. VMR steht für Video Mixing Renderer und die 7 bzw. 9 für die intern verwendete DirectX-Technologie. Der Filter VMR-7 ist standardmäßig in Windows XP enthalten. VMR-9 ist in der Installation von DirectX9 enthalten und kann somit auf allen System eingesetzt werden, die dieses Installiert haben. Abhängig von der Grafikkarte, wird die Verarbeitung in diesen Filter beschleunigt.

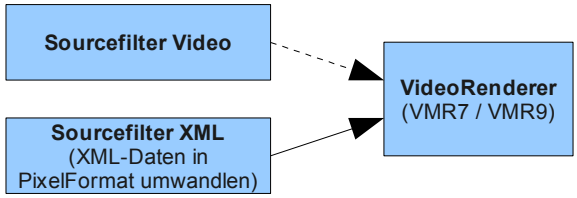


Abbildung 3: Mischung im VideoRenderer

4 Konzeptentwicklung

4.1 Ausgangssituation und Schwerpunkte

Ausgangspunkt für diese Konzeptentwicklung ist das bereits in der Einleitung genannte Szenario aus dem Bereich Videoanalyse im Sport. Zu einem Video mit Trainings- bzw. Wettkampfszenen werden vom Trainer Kommentare hinterlegt, welche sich der Sportler später mit dem Video zusammen ansehen kann. Dabei werden die Anmerkungen des Trainers direkt im Video dargestellt.

Dieses Anwendungsbeispiel soll allerdings nur den groben Leitfaden darstellen. Das ganze System sollte wenn möglich auch in anderen Anwendungsgebieten einsetzbar sein. So wäre es zum Beispiel denkbar, dass die Texte aus der XML-Datei mit den Zusatzinformationen auch außerhalb vom Video darstellbar sind. Um so etwas zu ermöglichen, muss das Schema für die XML-Datei möglichst flexibel gehalten werden.

Die Eingabe der Daten für die Zusatzinformation erfolgen über ein separates Programm, welchem das Schema der XML-Datei bekannt ist. Bei der Konzeptentwicklung wird ein Prototyp für solch ein Schema erarbeitet und vorgestellt.

Schwerpunkt dieser Arbeit ist allerdings das Konzept für die Wiedergabe der Videodatei und der dazugehörigen Zusatzinformationen. Bei diesen Zusatzinformationen wird es sich um einfachen Text handeln, so wie um einfache geometrische Figuren.

4.2 Konzept und Schema der XML-Datei

4.2.1 Die Dateiendung

Damit das Programm die Datei mit den Zusatzinformationen sofort erkennen kann, braucht diese eine unverwechselbare Dateiendung. Wenn man als englische Bezeichnung für diese Datei "ExtendedVideoInformation" annimmt, sind zwei Endungen möglich. Zum einen "evi" und zum anderen "xvi".

Um den allgemeinen Erwartungen bezüglich der Namensgebung gerecht zu werden, empfiehlt es sich für "Extended" den Buchstaben X zu nehmen. Er wird zum Beispiel auch für XML, eXtended Markup Language verwendet.

Um die Unverwechselbarkeit zu gewährleisten, hilft eine Suche auf verschieden Internetseiten die sich mit dem Thema Dateiendungen befassen. So liefert die Seite filext.com zum Beispiel für die Endung .evi nur einen Treffer.[13] Hierbei ist als Beschreibung für .evi folgendes Angegeben: "EchoView Index File". Bei der Software EchoView handelt es sich um ein Programm zur Auswertung von Fischschwärmen. Diese Dateiendung ist also schon vergeben, könnte aber trotzdem verwendet werden, da ein Konflikt eher unwahrscheinlich ist.

Bei der Suche nach .xvi auf der gleichen Internetseite und auch auf anderen Seiten, wie zum Beispiel endungen.de, wurden keine Treffer gefunden. Das lässt darauf schließen, das diese Dateiendung noch nicht vergeben ist und daher verwendet werden kann.

Somit ist festzuhalten, die Dateiendung für die Datei mit den Zusatzinformationen wird "xvi" sein.

13 http://filext.com/file-extension/evi [11]

4.2.2 Aufbau der XML-Datei

Um eine gültige XML-Datei zu erhalten, darf es nur ein Wurzelelement geben. Einfachheitshalber wird hier die Bezeichnung für den Dateityp dieser XML-Datei genommen, also <*extendedVideoInformation*>.

```
<extendedVideoInformation>
  <config>...</config>
  <common>...</common>
  <scenes>...</scenes>
</extendedVideoInformation>
```

Die Erste Ebene, also die Kindelemente des Wurzelknoten, unterteilt sich in drei große Bereiche. Ein Bereich für Einstellungen bezüglich der XML-Datei (<*config*>), ein Bereich für allgemeine Angaben zum Video (<*common*>), und schließlich ein Bereich für die Daten, die angezeigt werden sollen (<*scenes*>). Die Reihenfolge dieser Bereiche spielt dabei keine Rolle.

Der <config>-Bereich

Es ist immer Ratsam, bei dem Entwurf eines XML-Schemas einen Bereich für allgemeine Einstellung bezüglich der XML-Datei zu schaffen. Wichtigster Unterpunkt dabei ist die Version des verwendeten XML-Schemas. Mit Hilfe dieser Angabe, kann das Schema später erweitert und verändert werden. Es ermöglicht dem Programm eine Zuordnung der XML-Datei zu der verwendeten Schemaversion. (<*version*>)

Die erste wichtige Einstellung, für das Ausgangsszenario dieser Studienarbeit, ist der Name bzw. Pfad der Videodatei, für die die Zusatzinformation der XML-Datei gedacht sind. Hierfür steht das Element <*videofile*> zur Verfügung.

```
<config>
  <version>0.1</version>
  <videofile>...</videofile>
  <timeformat fps="25">
    frames
  </timeformat>
  <screen width="x" height="y"/>
</config>
```

Die zweite wichtige Einstellung ist das Zeitformat (<*timeformat*>). Wie bereits in Kapitel 3.3.2 dargestellt, wird es sich dabei um eine Ganzzahl handeln. Nun muss allerdings zu dieser Zahl noch eine Einheit definiert werden. Hier kann "nanosecond" als Wert verwendet werden, welches die Benutzung der Referenzzeit von 100 Nanosekunden angibt. Es kann aber auch "frames" als Wert des Elementes gesetzt werden. Wobei sich bei dieser Einstellung noch eine Erweiterung für das Element ergibt. Es muss angegeben werden wie viele Frames pro Sekunde angezeigt werden. Für diese Angabe erhält das Element ein Attribut "*fps*". In ihm kann eine Ganzzahl zwischen 25 und 50 eingetragen werden.

Für die spätere Darstellung der Inhalte, ist die Angabe der Videobreite (*"width"*) und Videohöhe (*"height"*) von besonderer Bedeutung (<*screen*>). Da es sich bei den Zusatz-

informationen um Vektordaten handelt, ist der Kontext, in dem sie stehen wichtig. Durch width und height wird die Größe der Zeichenfläche bestimmt. Somit können Größen- und Positionsangaben bei der Ausgabe der Zusatzdaten genau berechnet werden.

Der <common>-Bereich

Der Common-Bereich ist mehr allgemeiner Natur, hier werden zusätzliche Informationen über die Aufnahme (<*videofile*>) und über die Zusatzdaten (<*infofile*>) hinterlegt. Da dieser Bereich für diese Studienarbeit eher unwichtig ist, beschränken sich die Angaben auf den Autor der Zusatzdaten (<*author*>), bzw. den Kameramann der Videodatei (<*director*>) und auf eine kurze Beschreibung der beiden Dateien (<*description*>).

```
<common>
  <videofile>
    <director>...</director>
    <description>
      ...
    </description>
  </videofile>
  <infofile>
    <author>...</author>
    <description>
      ...
    </description>
  <infofile>
</common>
```

Der <scenes>-Bereich

Der Scenes-Bereich ist der wichtigste Abschnitt der XML-Datei, hier stehen die Informationen, die im Video eingeblendet werden. Wie bereits in Kapitel 3.3.2 erarbeitet, werden die einzelnen Szenen in einem eigenen Element zusammengefasst (<*scene*>). Für jede Szene muss eine Anfangszeit ("*timefrom*") und eine Endzeit ("*timeto*") als Attribut angegeben werden. Nur in dieser zeit werden die Zusatzdaten angezeigt.

```
<scenes>
  <scene timefrom="" timeto="">
    ...
  </scene>
  .
  .
  .
</scenes>
```

Die Reihenfolge der definierten Szenen spielt im XML nur eine geringe Rolle. Es ist ebenso möglich, dass sich Szenen überschneiden. Ist dies der Fall, werden die Informationen aus beiden Szenen angezeigt. Hierbei werden die Daten der jüngeren Szene über den alten Daten angezeigt. Haben die Szenen den gleichen Anfangszeitpunkt entscheidet die Reihenfolge in der XML-Datei. Der Einfachheit halber wird diese Variante gewählt, die Reihenfolge der Darstellung kann sich in späteren Schema-Versionen noch ändern. Der Inhalt dieser Szene-Elemente wird im nächsten Kapitel näher erläutert.

4.2.3 Speicherung der Zusatzinformationen für eine Szene

Die Speicherung der eigentlichen Informationen erfolgt ebenfalls in XML-Syntax. Um dabei möglichst flexibel zu sein, bietet es sich an hierfür eine Untermenge von SVG zu nutzen. Je nachdem wie stark man das ganze eingrenzt, kann man im eigentlichen Programm dann entscheiden, ob man einen eigenen Renderer schreibt oder einen vorhanden nutzt. Ratsam ist es es, erst einmal so weit wie möglich einzugrenzen und allmählich zu erweitern.

Grafische Objekte in der XML-Datei

Um die unterstütze Untermenge an SVG-Elementen zu Beginn der Entwicklung möglichst klein zu halten, wir die Möglichkeit an grafischen Primitiven auf die Elemente *rect*, *circle*, *ellipse* und *line*. Das Element *polyline* wird vorerst nicht unterstützt, da es sich notfalls durch *line* darstellen lässt. Ebenso wird das sehr mächtige *path*-Element nicht unterstützt, das es durch seine sehr hohe Komplexität den Aufwand für das Schreiben eines Beispielrenderers massiv in die Höhe treiben würde.

Für eine schnellere Verarbeitung der Daten wird das *style* und und das *transform* Attribut nicht unterstützt. Alle Wichtigen Eigenschaften aus style lassen sich als eigene Attribute darstellen, wie zum Beispiel fill und stroke oder stroke-width. Das transform-Attribute ist ähnlich wie das path-Element sehr umfangreich und würde daher den Rahmen dieser Studienarbeit sprengen.

Eine Auflistung der unterstützten grafischen Elemente und ihrer Attribute findet sich im Anhang Seite II.

Text in der XML-Datei

Für die Speicherung von Text wird, wie im SVG Standard definiert, das Element *text* verwendet. Allerdings ist innerhalb dieses Elementes nur Text erlaubt. Normaler weiße können hier Elemente wie zum Beispiel tspan verwendet werden, aber um das ganze einfach zu halten, entfällt dies. Daraus ergibt sich auch, dass der Text immer nur einzeilig sein kann.

Die Gestaltung des Texte ist bei der Prototyp Variante sehr eingeschränkt. Das text-Element in SVG bietet eine Vielzahl an Möglichkeiten und Wege den Text zu formatieren. Da dies aber ebenfalls den Rahmen dieser Studienarbeit sprengen würde, beschränkt es sich hier auf die Grundeigenschaften wie, Fontname, Fontgröße und Fontdicke. Die genauen angaben für die unterstützten Attribute finden sich im Anhang (Seite II).

4.3 Konzept Programmablauf

Der Programmablauf lässt sich am besten am Beispiel eines Testprogrammes erläutern. Das Programm selber muss eigentlich nicht wirklich viel machen. Es muss eine der beiden Dateien, das Video oder die XML-Datei öffnen und den Filtergraphen erstellen und verwalten. Um alle möglichen Testszenarien abzudecken, muss man das Video anhalten und sich darin vor und zurück bewegen können.

Datei öffnen

Alles fängt mit dem Öffnen einer Datei an. Wird die Videodatei geöffnet, muss nach der passenden XML-Datei gesucht werden. Hierbei wird, wie bereits in der Vorbetrachtung (Kapitel 3.2) beschrieben, nach einer Datei mit dem gleichen Dateinamen und der mittlerweile Definierten Endung ".xvi" gesucht. Wurde diese gefunden, wird überprüft, ob die Videodatei auch wirklich im config-Bereich dieser XML-Datei angegeben ist (XPath: "/extendedVideoInformation/config/videofile"). Ist dies nicht der Fall, wird ganz normal nur das Video wiedergegeben.

Wird die XML-Datei vom Programm geöffnet, wird nach der Videodatei gesucht, welche im config-Bereich angegeben ist. Existiert diese, kann mit dem Programmablauf fortgefahren werden.

Wenn dem Programm beides, die XML-Datei und die Videodatei bekannt sind, kann es den benötigten Filtergraphen erstellen. (siehe auch Anhang Seite III)

Filtergraph erstellen

Der Filtergraph besteht aus mindestens vier Filtern, meist kommen allerdings noch weitere Filter für die Verarbeitung der Video- und Audiodaten hinzu. Grundbestandteil des Filtergraphen sind der Source-Filter für die Videodatei, der Source-Filter für die XML-Datei, welcher allerdings noch entwickelt werden muss, sowie der Overly-Mixer-Filter und der VideoRenderer-Filter.

Der einfachste Weg den Filtergrafen zu erstellen, besteht darin, erst mal nur den Graphen für die Videodatei zu erstellen. Anschließend wird der VideoRenderer-Filter wieder abgetrennt und an seine stelle der Overlay-Mixer-Filter gesetzt. Der VideoRenderer-Filter wird nun mit dem Ausgang des Overlay-Filter verbunden. Als letztes wird der Sourcefilter für die XML-Datei in den Filtergraphen eingebracht. Dieser wird nun mit dem zweiten Eingang des Overlay-Filters verbunden. Nun kann der Filtergraph verwendet werden.

Wiedergabe des Videos

Für die Wiedergabe des Videos stehen dem Programm jetzt nur noch die Funktionen des Filtergraphen bzw. dessen Interface zur Verfügung. Die Einblendung der Zusatzinformationen wird jetzt vollautomatisch durch die entsprechenden Filter im Graphen geregelt. Die Aktionen Play, Stop, Pause, Vorspulen, Zurückspulen und schneller bzw. langsamer Abspielen werden durch die interne Kommunikation im Filtergraphen an alle Filter weitergeleitet und erfordert daher kein weiteres Eingreifen in das System.

Die Einblendung der Zusatzinformationen hängt jetzt nur noch von dem Source-Filter der XML-Datei ab. Dieser wird im nächsten Kapitel näher erläutert.

4.4 Konzept Source-Filter für XML-Datei

4.4.1 Allgemeine interne Verarbeitung

Der Source-Filter der XML-Datei sorgt für die Übersetzung der XML-Daten in ein Grafikformat, welches in das Video eingeblendet werden kann. Darüber hinaus werden nur die Daten der aktuellen Szene verarbeitet und ausgegeben.

Als erstes muss geklärt werden, welches Format die Grafikdaten nun eigentlich haben müssen. Ein Blick in die Dokumentation des Overlay-Mixer-Filter gibt für Eingang-Pins größer null, Pin null ist das Video, das Format "Microsoft DirectDraw RGB" (MEDIASUBTYPE_RGBxx) an.[14] Daraus ergibt sich, das der Source-Filter eine DirectDraw Surface ausgeben muss, sonst können die zwei Filter nicht verbunden werden.

Da das Umwandeln der XML-Daten in das entsprechende Grafikformat relativ viel Zeit in Anspruch nehmen kann, empfiehlt es sich dies Vorausschauend zu tun. Das heißt, das man eine gewisse Anzahl gerenderten Daten bereits vorhält. Also das ein Overlay, was erst in der nächsten oder übernächsten Szene eingeblendet wird, bereits gerendert ist. Wärend der Filter die Daten der aktuellen Szene ausgibt, können bereits die Daten für die nächste Szene gerendert werden.

Für das effektive Arbeiten des Filters ist es also erforderlich, alle Anfangs und Endzeiten zu kennen. So weiß der Filter, was als nächstes gerendert werden muss, eh er es ausgeben muss. Daraus ergibt sich, das der Filter, wenn er erzeugt wird, alle Anfangs und Endzeiten erfassen muss. Diese können geordnet in einer Liste gespeichert werden und der ein Listenzeiger zeigt immer auf die aktuelle Ausgabe.

Damit der Speicher nicht irgendwann überläuft, müssen die schon angezeigten gerenderten Zusatzdaten auch wieder aus dem Speicher entfernt werden. Diese drei Aufgaben, Ausgabe, Rendern und Löschen lassen sich gut in drei separate Threads auslagern, da sie kaum Berührungspunkte haben und gleichzeit geschehen können. Eine Erläuterung zu diesen Threads findet sich in den anschließenden drei Kapiteln.

14 nach MSDN: Overlay Mixer Filter [12]

4.4.2 Der Ausgabethread / Der Hauptthread

Da die Ausgabe der Grafikdaten eng an die Kommunikation zwischen den Filtern gebunden ist, ist der Hauptthread für die Ausgabe zuständig.

Anhand der bereits im vorangegangenen Kapitel genannten Liste von Anfangs und Endzeiten weiß der Filter, was ausgegeben werden muss. Der aktuellen Listeneintrag muss einen Zeiger auf eine DirectDraw Surface enthalten, welche die gerenderten Zusatzinformationen für die aktuelle Szene enthält. Diese Surface muss dann über den Ausgangspin des Filters an den nächsten Filter im Graphen, den Overlay-Mixing-Filter, weitergegeben werden.

Der Hauptthread ist für die Erzeugung dieser Liste und Positionierung des Listenzeigers verantwortlich. Er muss bei Erzeugung des Filters dafür sorgen, das die Liste mit den benötigten Informationen gefüllt wird. Er muss den Listenzeiger beim Abspielen des Vidoes weiter rücken und er muss bei einem Sprung im Video die entsprechende Stelle in der Liste suchen und den Listenzeiger auf diese setzen.

Liste mit Start- und Endzeiten erzeugen

Bei der Erzeugung der Liste muss besonders auf Überschneidungen geachtet werden. Sind die Szenen immer getrennt von einander, ist die Sache ganz einfach. Es wird ein Eintrag in die Liste mit der Startzeit eingefügt und es wird vermerkt, das dafür eine Surface erstellt werden muss. Als nächster Eintrag wird die Endzeit eingefügt, zusätzlich wird vermerkt, das keine Daten ausgegeben werden müssen.

Liegt nun allerdings die Endzeit der einen Szene nach der Anfangszeit der folgenden Szene gibt es eine Überschneidung. Ist dies der Fall, müssen für die Folgeszene zwei Surfaces erstellt werden, einmal mit den Daten aus beiden Szenen und einmal nur mit den Daten der Folgeszene. Ein Beispiel dafür findet sich in den Anlagen (Seite IV).

4.4.3 Der Render-Thread

Der Render-Thread ist, wie der Name schon sagt, für das Rendern der XML-Daten zuständig.

Durch den aktuellen Listenzeiger weiß er, welche Daten als nächstes gerendert werden müssen. Es müssen die nächsten drei bis fünf Surfaces gerendert sein. Ist dies nicht der Fall, geht der Render-Thread chronologisch, vom aktuellen Listenzeiger aus, durch die Liste und rendert die XML-Daten. Er leistet quasi die Vorarbeit für die Ausgabe. Nur was er vollständig gerendert hat, kann auch ausgegeben werden.

Kritische Punkte für seine Vorarbeit sind der Einstiegspunkt und ein Sprung im Video, diese machen seine bereits geleistete Arbeit Großteils wertlos. Es muss so schnell wie möglich die Surface für die aktuelle Wiedergabe gerendert werden. Geschieht dies nicht schnell genug, kann es im schlimmsten Fall zur permanenten Verzögerung bei der Darstellung der Zusatzinformationen kommen. In den meistens Fällen wird sich aber nur die Einblendung der aktuellen Zusatzinformationen verzögern.

Das Rendern

Ehe etwas gerendert werden kann, muss eine DirectDraw Surface erzeugt werden, auf die die Ausgabe erfolgen kann. Der Hintergrund der Surface muss eine Farbe sein, die in den XML-Daten nicht vorkommt, da sie bei der Ausgabe der Daten transparent wird. Nur so ist es möglich, das man das Video noch erkennt.

Das Rendern der, für diese Arbeit definierten, SVG-Elemente, sollte kein Problem darstellen, da es zu jedem Element eine entsprechende Zeichenfunktion in DirectX gibt. Passender weiße haben die Funktionen auch noch die gleichen Parameter, wie die SVG-Elemente Attribute haben. So wird für das Zeichnen eines Kreises (<circle>) zum Beispiel die Funktion "DrawCircle" aufgerufen.

4.4.4 Der Lösch-Thread

Dieser Thread ist einzig und allein für das Aufräumen der verwendeten Resourcen zuständig. Er sorgt dafür, das keine Speicherlöscher entstehen und der Verbrauch an Arbeitsspeicher nicht ständig ansteigt.

Der Thread merkt sich den Listenzeiger. Wenn dieser sich ändert und zum nächsten Listeneintrag springt, kann die gerade verwendete Surface gelöscht bzw. freigegeben werden. Da die Wiedergabe des Videos eine Vorwärtsbewegung des Listenzeigers bedingt, können die Surfaces der vorhergehenden Einträge freigegeben werden.

Kritisch wird es bei einem Sprung im Video, da jetzt mehrere Surfaces freigegeben werden müssen. Der Render-Thread hat bereits mehrere Surfaces an der alten Position erzeugt, diese müssen nun entfernt werden. Da der Lösch-Thread noch die alte Position gespeichert hat, sollte dies kein Problem sein.

4.4.5 Kritische Abschnitte bei der Arbeit mit den Threads

Bei der Arbeit mit den Threads besteht eine gemeinsamme Komponente und zwar die Liste mit den Start- und Endzeiten, sowie den Surfaces.

Es muss darauf geachtet werden, das eine Surface erst ausgegeben werden kann, wenn diese fertig gerendert ist. Außerdem darf eine Surface nur gelöscht werden, wenn sie nicht gerade ausgegeben wird.

Um hier Fehler zu vermeiden, müssen die entsprechenden Resourcen geschützt werden. Die einfachste Variante, um dafür zu sorgen, dass eine Surface erst ausgegeben werden kann, wenn sie fertig gerendert ist, besteht darin, den Zeiger auf die Surface erst dann der Liste bekannt zu geben.

Um zu verhindern, das eine Surface gelöscht wird, obwohl sie gerade ausgegeben wird, besteht in einer Markierung, das diese Surface sich in Verwendung befindet.

5 Zusammenfassung und Ausblicke

Das in dieser Studienarbeit erarbeitete Konzept ist in den gestalterichen Möglichkeiten stark eingeschränkt. Da das Konzept aber so entwickelt wurde, das es nach oben hin offen ist, kann es ohne Probleme erweitert werden.

Durch die Verwendung des SVG-Schemas ist es ebenso möglich ohne Probleme einen anderen Renderer einsetzen kann. Dies wird vor allem durch die Modularisierung des Source-Filters der XML-Datei begünstigt.

Eine mögliche Erweiterung des Source-Filters besteht im Ausbau der Surface-Verwalltung. Damit nicht ständig neue Surfaces erstellt werden müssen, bzw. freigegeben werden müssen, kann man das Programm auf Verwendung von 5 oder mehr, ständigen verfügbaren, Surfaces einschränken. Wenn eine Surface nicht mehr benötigt wird, wird auf ihr der Inhalt einer anderen Szene gerendert. So kann unter anderem auch die Bearbeitungsgeschwindigkeit des Render-Threads verbessert werden.

Festzuhalten ist, das ein Konzept für die Entwicklung eines Systems zur dynamischen Einblendung von Grafiken und Text in ein Video, entwickelt wurde. Dieses Konzept kann nun Implementiert werden.

Abkürzungszeichens

API	Application Programming Interface (engl. Programmierschnittstelle)
ASF	Advanced Systems Format - Conatainerformat für Audio und Video
AVI	Audio Video Interleaved - Containerformat für Video
CAD	Computer Aided Design
COM	Component Object Model
DLS2	Downloadable Sounds Level 2 - Klangerzeugung aus Wave-Samples
DTD	Document Type Definition - für Schema Definition
DV	Digital Video
MIDI	Musical Instrument Digital Interface - Dateiformat
MP3	MPEG-1 Layer 3
MPEG	Moving Picture Experts Group, meist aber Synonym für einen Videostandard
RSS	Really Simple Syndication - Format für Newsfeeds
SDK	Software Developer Kit
SMIL	Synchronized Multimedia Integration Language
SOAP	Simple Object Access Protocol - Kommunikationsprotokoll
SVG	Scaleable Vector Graphic
VMR	Video Mixing Renderer
W3C	World Wide Web Consortium - Gremium zur Standardisierung im WWW
WAV	Wave Dateiformat zur digitalen Speicherung von Audiodaten
WMA	Windows Media Audio
WMV	Windows Media Video
XHTML	Extensible HyperText Markup Language - Sprache für Webseiten
XML	Extensible Markup Language - Metasprache
XSD	XML-Schema-Definition
XSLT	Extensible Stylesheet Language Transformation
XVI	Extended Video Information - Format für Zusatzinformationen

Literaturverzeichnis

[1] Wikipedia: DirectX; http://de.wikipedia.org/wiki/DirectX

[2] MSDN: Introduction to DirectShow; http://msdn2.microsoft.com/en-us/library/ms786508(VS.85).aspx

[3] Wikipedia: DirectShow; http://en.wikipedia.org/wiki/DirectShow

[4] Pesce, Mark D.; Programming Microsoft DirectShow for Digital Video and Television; Microsoft Press; 2003

[5] MSDN: About DirectShow Filters; http://msdn2.microsoft.com/en-us/library/ms778825(VS.85).aspx

[6] MSDN: Reference Clocks; http://msdn2.microsoft.com/en-us/library/ms787549(VS.85).aspx

[7] Wikipedia: Extensible Markup Language; http://de.wikipedia.org/wiki/Xml

[8] Wikipedia: XSL Transformation; http://de.wikipedia.org/wiki/Xslt

[9] Wikipedia: Scalable Vector Graphics; http://de.wikipedia.org/wiki/SVG

[10] Wikipedia: XML Schema; http://de.wikipedia.org/wiki/XML_Schema

[11] FileExt.com: File Extension .EVI Detail; http://filext.com/file-extension/evi

[12] MSDN: Overlay Mixer FIlter; http://msdn2.microsoft.com/en-us/library/ms787455(VS.85).aspx

Abbildungsverzeichnis

Anlagen

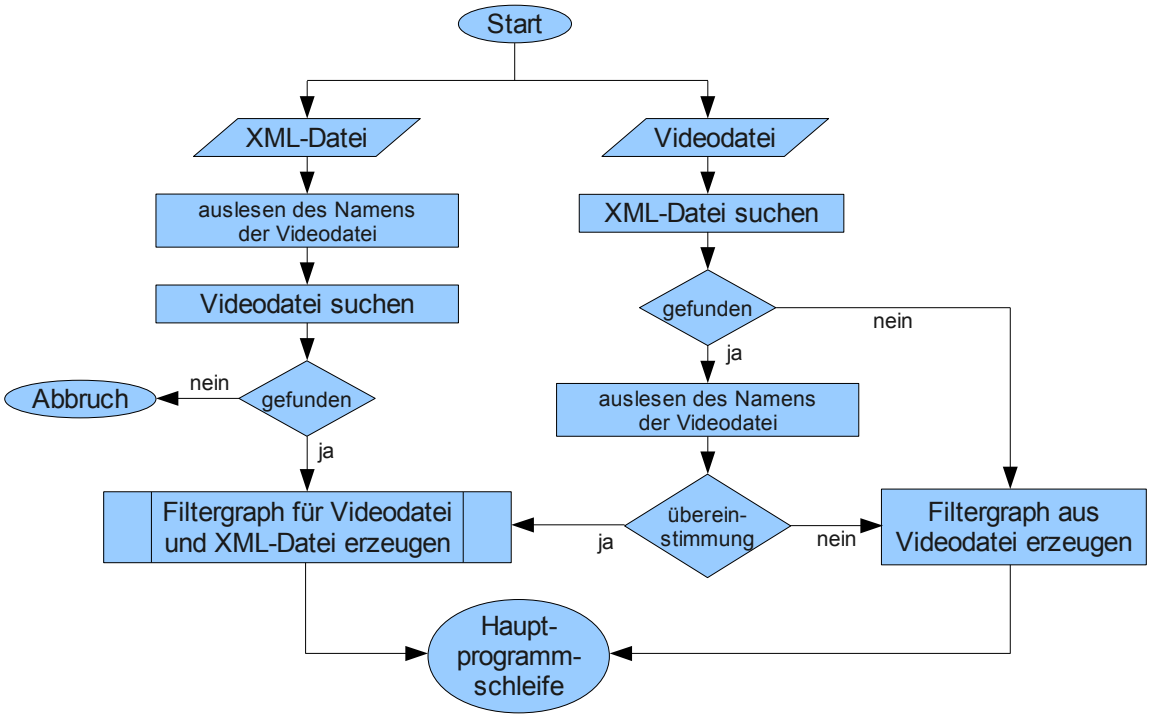
Codebeispiel: ein einfaches XSLT-Stylesheet

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    exclude-result-prefixes="html">
    <xsl:output method="xml" />

    <!-- Einstiegspunkt -->
    <xsl:template match="/">
        <xsl:apply-templates select="titel"/>
        <xsl:apply-templates select="text"/>
    </xsl:template>

    <!-- 'title'-Tag verarbeiten -->
    <xsl:template match="titel">
        <h1>
            <xsl:value-of select="."/>
        </h1>
    </xsl:template>

    <!-- Eine Variable -->
    <xsl:variable name="pretext" value="'Text: '"/>

    <!-- 'text'-Tag verarbeiten: -->
    <xsl:template match="text">
        <!-- Inhalt der Variable ausgeben -->
        <xsl:value-of select="$pretext"/>
        <br />
        <!-- nur ausgeben, wenn mehr als 5 Zeichen im Tag -->
        <xsl:if test="string-length(.) &gt; 5">
            <xsl:value-of select="."/>
        </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

Elemente für das XML

```
<rect>
      x/y               -> Position Linke Obere Ecke
      width/height      -> Höhe und Breite
      rx/ry             -> Abrundung der Ecken (optional, 0)
      fill              -> Füllfarbe (optional, weiß)
      stroke            -> Farbe der Umrandung (optional, schwarz)
      stroke-width      -> Breite der Umrandung (optional, 1)

<circle>
      cx/cy             -> Position Mittelpunkt
      r                 -> Radius
      fill/stroke/stroke-width -> siehe <rect>

<ellipse>
      cy/cy             -> Position Mittelpunkt
      rx/ry             -> Breite / Höhe
      fill/stroke/stroke-width -> siehe <rect>

<line>
      x1/y1             -> Punkt von (Startpunkt)
      x2/y2             -> Punkt nach (Endpunkt)
      stroke/stroke-width -> siehe <rect>

<text>
      x/y          -> Textposition
      font-size    -> Größe der Schriftart
      font-weight -> Dicke des Buchstaben = normal|bold|bolder|lighter
                     or 100|200|300|400|500|600|700|800|900
                     (optional, normal)
      font-style   -> normal|italic (optional, normal)
      text-anchor -> start|middle|end = Textbündigkeit bezogen auf x
                     (optional, start)
```

Programmablauf: Datei öffnen



Beispiel: Liste mit Start- und Endzeiten erstellen

```
<scenes>
  <scene timefrom="0" timeto="5">...</scene>          // tag1
  <scene timefrom="10" timeto="20">...</scene>        // tag2
  <scene timefrom="15" timeto="25">...</scene>        // tag3
</szenes>
```

Zeitangabe	Surface erzeugen	XML-Tag Nr.	Surface
0	ja	1	xxx
5	nein	1	0
10	ja	2	xxx
15	ja	2+3	xxx
20	ja	3	xxx
25	nein	3	0