Seminar-Arbeit

XML 3D Formate - X3D und XVL

Michael Bayer

Seminar-Arbeit: XML 3D Formate - X3D und XVL

by Michael Bayer

Table of Contents

Vorwort

Immer mehr Anwendungsgebiete erfordern dreidimensionale Visualisierung von Daten, Strukturen und Objekten. Dies macht auch ein standardisiertes und allgemein anerkanntes Datenaustausch-Format erforderlich um eine störungsfreie Interaktion zwischen verschiedenen Applikationen zu gewährleisten.

Ein solches Format muss einige Anforderungen erfüllen:

- Es muss offen und unabhängig spezifiziert sein um Abhängigkeiten von einzelnen Firmen zu verhindern.
- Es ist muss "Internet-fähig" sein, d.h. es muss in Standardbrowsern angezeigt werden können.
- Weiterhin ist es nötig dass das Format erweiterbar ist um mit den schnellen Enwicklungszyklen des 3D-Bereiches schritt halten zu können.

Aufgrund dieser Anforderungen drängt sich als Basis dieses Formates XML auf.

Diese Ausarbeitung soll neben der Erläuterung der Grundlagen dieses Gebietes zwei solcher Formate kurz vorstellen und ihre Vor- und Nachteile erörtern.

Chapter 1. Warum XML

XML (eXtensible Markup Language) ist ein SGML Dialekt und folglich eine Auszeichnungssprache welche auf Tags basiert die Informationen einschließen. Anders als zum Beispiel HTML welches ebenfalls ein SGML Dialekt ist ist XML deutlich mächtiger da keine semantischen sondern nur syntaktische Merkmale definiert werden. Dies macht XML zu einer Art Metasprache welche verwandt werden kann um verschiedenste Dokumenten-Typen zu definieren.

Ziele von XML:

- XML soll in gesamten Web verwendet werden können.
- XML soll eine große Anzahl von Applikationen unterstützen.
- XML soll auch mit SGML kompatibel sein.
- Programme, die XML tauglich sind, sollen leicht zu entwickeln sein.
- Die Zahl der Zusatz-Merkmale von XML soll ein absolutes Minimum betragen - idealerweise Null.
- XML Dokumente sollten einen "menschlichen Maßstab" haben.
- XML Dokumente sollten leicht erstellt werden können.

XML ist ein reines Text-Format. Das bringt einige Vorteile mit sich, zuvorderst die Einfachheit mit der XML-Dokumente editiert werden können. Dies löst das bekannte Henne-Ei-Problem dem man oft beim Entwickeln von Datenformaten begegnet: Ohne Test-Dateien kann kein Parser geschrieben werden, und ohne die Daten die der Parser liefert kann keine Test-Datei generiert werden. Auch sorgt diese Eigenschaft dafür dass die Programm Daten und somit die Qualität des Parsers sehr einfach vom Menschen überprüft werden kann.

XML ist auf allen gängigen Betriebssystemen mit Bordmitteln zu parsen, beziehungsweise es steht eine Vielzahl von System-Bibliotheken zur Verfügung die hierzu herangezogen werden können.

Ein weiterer Vorteil von auf XML basierenden Datenformaten besteht darin dass sie ein nahtloses Einbetten von bestehenden XML-Formaten ermöglichen. Dies ist trivial ersichtlich da jedes einzelne Datenformat offensichtlich ebenfalls XML-konform ist.

Aber: XML birgt auch Nachteile. Das Parsen zum Beispiel ist zwar einfach zu implementieren, aber die Eigenschaft ein reines Text-basierendes Format zu sein bringt mit sich dass eben dieses nur vergleichsweise langsam zu bewerkstelligen ist, da oft Zeichenketten zu binär-Daten konvertiert werden müssen. Auch führt die Erweiterbarkeit des Formates sehr oft dazu dass Entwickler beim Spezifizieren solcher XML-Formate dazu tendieren diese oft nicht ausreichend oder zu ungenau zu Definieren.

All dies macht XML zu einer idealen Sprache um Datenaustausch-Formate zu definieren. Der Nachteil des langsamen Parsens kommt hier nicht so stark zum Tragen da diese Formate oft nur als Zwischenstufe

eben zum Datenaustausch zwischen verschiedenen Applikationen genutzt werden.

Chapter 2. Repräsentation

Die Objektrepräsentation im dreidimensionalen Raum ist ein im Feld der Computergrafik ein viel-diskutiertes Thema für welches sich keine allgemein-gültige Lösung herausgestellt hat. Es werden verschiedene Ansätze verfolgt, von denen jeder eigene Vor- und Nachteile hat. Einige dieser Ansätze sollen hier kurz diskutiert werden.

Die geometrischen Daten der in 3D-Formaten abzuspeichernden Objekte müssen in geeigneten Datenstrukturen abgelegt werden. Die Wahl einer geeigneten Objektrepräsentation ist keine einfache Aufgabe. Die Darstellungsform wird sowohl durch die Rendering- Technik, d.h. die Art der Erzeugung eines Bildes, als auch durch die Anwendung bestimmt. Eine Vielzahl von Aspekten können bzw. müssen berücksichtigt werden. Zu nennen sind z.B. geringer Speicherbedarf, schnelle Darstellbarkeit, die Durchführbarkeit linearer Transformationen und Kombinationsoperatoren, die Möglichkeit der Interaktion und die Möglichkeit der automatischen Generierung der Datenstruktur anhand von digitalisierten realen Objekten (z.B. mittels 3D-Scanner oder CT).

Im folgenden sollen einige verbreitete Repräsentationsschemata mit ihren jeweiligen Vor- und Nachteilen vorgestellt werden - die Punkt-Wolke, das Draht-Modell sowie Flächen- und Volumen-Modelle.

2.1. Punkt-Wolke

Die Punkt-Wolke (engl. Scatter Plot) ist die einfachste - allerdings auch die ungenaueste - Möglichkeit Geometrie im dreidimensionalen Raum zu repräsentieren. Die Darstellung des Objektes erfolgt durch eine Menge von Punkten - Vertices genannt. Es bestehen keine Relationen zwischen Punkten und somit kann diese Form der Darstellung keinerlei Informationen über Kanten, Volumen oder Oberflächen des Objektes tragen, jedoch ist es möglich solche Daten heuristisch zu erzeugen, zum Beispiel indem man Polygone über benachbarte Vertices aufspannt. Dieses Vorgehen führt jedoch nicht oft zu befriedigenden Ergebnissen.

Die Objekt-Struktur der Punkt-Wolke ist sehr einfach: einzige atomare Strukturen sind die Punkte im Raum, Objekte bestehen aus Mengen von Punkten. Bei Punkt-Wolken ist der Interpretationsspielraum des Betrachters in der Regel sehr hoch. Um gekrümmte Kanten und Oberflächen zu repräsentieren, benötigt man sehr viele Punkte, was zu einem entsprechend hohen Speicherplatz-Verbrauch führt. Die Vorteile der Punkt-Wolke liegen bei angemessener Verwendung im geringen Speicherplatz und schnellen und einfachen Transformationen.

Da Punkt-Wolken einfach mittels 3D-Scanner erzeugt werden können kommen sie oft als Zwischenergebnis nach zum Einsatz.

Abb. 2 - Ergebnis des 3D-Scans eines Pokals als Punkt-Wolke

2.2. Draht-Modell

Beim Draht-Modell (engl. Wireframe Modell) beschränkt sich die Repräsentation auf Kanten zwischen Punkten (Vertices). Oft wird davon ausgegangen dass nur gerade Kanten möglich sind, man kann aber durchaus auch Kurven, Bögen etc. zulassen, weil dies aber die Darstellung sehr kompliziert werden lässt wird diese Möglichkeit im Allgemeinen nicht zugelassen.

Prinzipiell kann man als Struktur in Draht-Modellen jegliche Art von Polygonen verwenden, aber da sich jegliches Polygon aus Dreiecken zusammensetzen lässt wird im Allgemeinen dieses als atomares Primitiv verwandt. Daraus ergeben sich mehrere Möglichkeiten die Kanten-Informationen in Draht-Modellen zu speichern.

Die intuitivste Methode ist die einzelnen Vertices einfach in Dreiergruppen in einer Liste zu speichern. Dabei ergibt sich allerdings das Problem dass Vertices - da sie im Allgemeinen nicht nur Eckpunkte eines Dreiecks sind - mehrfach abgespeichert werden müssen. Das lässt sich lösen indem man eine Liste von Vertices (Vertex Array) und eine Liste von Index-Tupeln welche dann die Kanten definieren, aber auch hier besteht Redundanz, und zwar werden Indices mehrfach gespeichert. Deswegen werden

Draht-Modelle oft als Dreiecks-Netze (Triangle Mesh, Triangle List) gespeichert, was bedeutet dass man eine Index-Liste in der Art speichert so dass drei aufeinander folgende Einträge in dieser ein Dreieck ergeben. Redundanz noch mehr verringern lässt sich durch das Speichern als Dreiecksstreifen (Triangle Strip). Hierbei werden die Indices in einer geeigneten Reihenfolge gespeichert so dass drei beliebige Einträge ein Dreieck ergeben. Eine letzte Methode mit noch weniger Einträgen in der Index-Liste auszukommen ist das Speichern als Dreiecks-Fächer, wobei ausgehend von einem Startpunkt nur noch zwei Einträge nötig sind ein Dreieck aufzuspannen (vgl. Abb. 3).

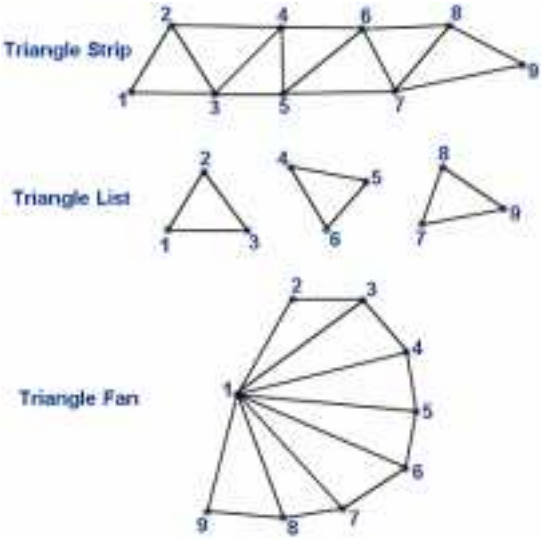

Abb. 3 - Verschiedene Methoden Dreiecks-Modelle zu speichern

2.3. Flächen-Modell

Um dreidimensionale Objekte realistischer und in richtiger Sichtbarkeit darstellen zu können, braucht man zumindest die Information, zwischen welchen Kanten sich Flächen befinden. Meistens wird dabei so vorgegangen, dass jedes Objekt direkt aus den das Objekt begrenzenden Flächen konstruiert wird; man spricht daher von einem Flächen-Modell.

Zur Modellierung der Flächen können verschiedene Grundelemente wie Polygone, Kreis- oder Ellipsen-Flächen oder mathematisch definierte Flächen verwendet werden. Flächen-Modelle werden häufig dort eingesetzt, wo komplexe, mehrfach gekrümmte Oberflächen modelliert und bearbeitet werden sollen, wie im Flugzeug- und Automobilbau. Da beim Flächen-Modell die Objekte nur aus Oberflächen bestehen und daher kein Inneres haben, treten verschiedene Probleme auf, z. B. bei der Durchführung von Schnitten. Teilweise transparente Objekte ohne definierte Oberflächen wie Wolken, Explosionen oder Feuer sind damit nur über Tricks simulierbar. In diesem Bereich kann ein anderes Prinzip, die Volumen-Grafik, seine Stärken ausspielen (vgl. Kapitel 2.4). Im Bereich der Flächen-Modelle sind besonders zwei Prinzipien weit verbreitet, Polygon-Netze und bikubische Parameter-Oberflächen.

2.3.1. Polygon-Netze

Polygon-Netze sind bei weitem das weit-verbreitetste Flächen-Modell. Dabei ist der Schritt vom Drahtgitter-Modell zum Polygon-Netz ein sehr kleiner, da er eigentlich nur ein Frage der Darstellung, nicht der Speicherung ist. Deshalb gelten auch hier die selben Voraussetzungen wie bei Drahtgitter-Modellen und man kann die selben Techniken anwenden.

Diese Methode Flächen-Modelle zu speichern ist vor allem deshalb so beliebt weil hierbei Tests auf zum Beispiel Konsistenz und Integrität sehr einfach zu implementieren sind. Auch hat sich gezeigt dass Dreiecke als atomare Einheiten für moderne Grafik-Hardware sehr günstig darzustellen sind. Allerdings werden zum Darstellen solcher Flächen-Modelle Verfahren benötigt die nicht sichtbare Linien und Oberflächen vor dem eigentlichen Zeichnen heraus filtern. Wie beim Draht-Modell ist auch beim Polygon-Netz die Darstellung gekrümmter Oberflächen problematisch. Sie müssen durch planare Polygon-Flächen approximiert werden, was bei guter Näherung zu einer unvertretbar hohen Zahl von Polygonen führen kann. Trotzdem überwiegen die Einfachheit der angesprochenen Tests sowie die effektive Algorithmen und Hardware zur Darstellung.

2.3.2. Parameter-Oberflächen

Im Unterschied zur Darstellung durch Polygon-Netze erfolgt die Abbildung eines dreidimensionalen Objekts bei den bikubischen Parameter-Oberflächen (auch: Freiformfläche) durch gekrümmte Oberflächen, die als Patches bezeichnet werden.

Jeder Patch wird durch mathematische Formeln definiert, welche die Position im dreidimensionalen Raum sowie die Form des Patches bestimmen. Die Formeln ermöglichen es, jeden Punkt auf der Oberfläche eines Patches zu erzeugen. Form oder Krümmung eines Patches können durch eine Bearbeitung der mathematischen Beschreibung modifiziert werden, woraus sich weitreichende Möglichkeiten zur Interaktion ergeben.

Dennoch gibt es auch bei den Freiformflächen bedeutende Probleme: Es ist sehr aufwendig, die Patches direkt zu rendern oder zu visualisieren. Wenn die Form eines einzelnen Patches in einem Netz von Patches verändert wird, entstehen Probleme, die glatten Übergänge zwischen dem Patch und seinen Nachbarn zu bewahren. Bikubische parametrische Patches können entweder eine exakte oder eine angenäherte Darstellung sein. Sie können nur eine exakte Darstellung von sich selbst sein, was bedeutet, dass jedes Objekt z.B. ein Karosserie-Blech nur dann genau dargestellt werden kann, wenn seine Form exakt der Form des Patches entspricht.

Um solche Parameter-Oberflächen nun zu rendern muss es in ein Polygon-Netz umgewandelt werden. Dies geschieht durch eine Technik die Subdivisioning genannt wird, wobei die Flächen iterativ durch Dreiecke angenähert werden. Es ist leicht ersichtlich dass dies je nach angestrebter Genauigkeit sehr schnell zu sehr großen Datenmengen führen kann.

Die Erstellung und Reproduktion von Kurven und Oberflächen findet vielseitig Verwendung,

insbesondere im Karosseriebau. Ziel hierbei ist mit möglichst wenig Informationen über die Kurve (Fläche) eine hohe Flexibilität und vor allem vollständige Rekonstruierbarkeit zu erreichen. Man kann solche Kurven mittels parametrischer Funktionen auf verschiedene Weise erzeugen, z.B über Polynominterpolationen. Eine parametrisierte Kurve im Raum hat die folgende Form:

f: [0,1] -> (f(u), g(u), h(u))

Wobei f, g und h reelle Funktionen sind. Somit bildet f(u) die reelle Zahl u aus dem Intervall [0, 1] auf einen Punkt im Raum ab.

Ein Spline ist eine beliebig glatte Kurve welche über ihre Kontrollpunkte (Control Vertices) definiert wird. Splines haben einen Anfang und ein Ende, und somit eine Richtung welche auch umgekehrt werden kann. Ein Basis-Spline (B-Spline) ist eine aus mehreren Segmenten zusammengesetzte Spline. Die Bezierkurve kann als Spezialfall der B-Splines angesehen werden. Somit sind B-Splines eine einfache Möglichkeit eine durch drei oder mehr Punkte interpolierte Kurve zu definieren. Der Name Basis-Spline rührt daher dass diese Art der Kurve durch sogenannte Basis-Funktionen erzeugt wird. Eine weitere Besonderheit der B-Splines besteht darin dass sich Veränderungen an den Kontrollpunkten nur auf den lokalen Abschnitt der Kurve auswirken.

Bezierkurven wurden speziell für den Computergestützten Entwurf von Karosserien entwickelt. Sie sind ein Basis-Spline welches aus nur einem Segment besteht und von zwei Kontrollpunkten gesteuert wird. Zu diesen Kontrollpunkten wird zusätzlich noch ein Vektor assoziiert wessen Länge und Richtung den Einfluss des Control Vertex auf die Kurve verändert.

NURBS steht für Non-Uniform Rational Basis-Spline. Es ist eine mathematische Definition von Kurven, Flächen und Volumen-Körpern und hat sich als Industriestandard für das Entwerfen und Modellieren etabliert. Non-Uniform bedeutet dass der Einflussbereich eines Kontrollpunktes variieren kann was besonders beim Modellieren unregelmäßiger Flächen hilfreich ist. Rational bedeutet dass die definierende Gleichung nicht durch ein einzelnes Summenpolynom sondern vielmehr durch das Verhältnis zweier Polynome definiert ist.

Da der Übergang von NURBS-Kurven zu -Flächen trivial ist und im allgemeinen die gleichen Voraussetzungen für beides gelten wird im weiteren nur noch von Kurven die Rede sein. Es existieren zwei Arten von NURBS-Kurven:

- Punkt-Kurven zeichnen sich dadurch aus dass ihre Kontrollpunkte derart eingeschränkt sind dass diese sich auf der Kurve befinden und
- CV-Kurven welche über Steuer-Scheitelpunkte definiert werden die nicht notwendigerweise auf der Kurve sondern auf einem Steuer-Gitter liegen.

NURBS-Kurven werden im Wesentlichen von drei Dingen beeinflusst:

- Einer Liste von Kontrollpunkten welchen jeweils eine Wichtung zugeordnet wird welche das Maß des Einflusses des Kontrollpunktes auf die Kurve angibt - ist diese Wichtung für alle Punkte gleich ergibt sich eine rationale Kurve. Kontrollpunkte welche darauf beschränkt sind auf der Kurve zu liegen werden Bearbeitungspunkte genannt und haben trivialerweise keine Wichtung. Kontrollpunkte sind Koeffizienten der NURBS-Basis-Funktionen und ihre Veränderungen wirken sich nicht über die benachbarten Control Vertices hinaus aus. Weiterhin ist die Auswirkung der Kontrollpunkte auf die Kurve kumulativ, sprich mehrere übereinander-liegende Kontrollpunkte verschärfen die Krümmung der Kurve.
- Der Grad eine NURBS-Kurve wird als positive Ganzzahl angegeben welche meist aus [1, 5] stammt. Die Ordnung einer Kurve wird als Grad+1 definiert. Der Grad einer Kurve ist der höchste Exponent in der zum Darstellen der Kurve verwendeten Gleichung. Eine lineare Gleichung ist eine Gleichung ersten Grades, eine quadratische Gleichung ist eine Gleichung zweiten Grades. NURBS-Kurven werden im Allgemeinen durch kubische Gleichungen repräsentiert und sind somit dritten Grades. Höhere Grade sind möglich, aber normalerweise nicht nötig.
- Da sie mathematisch generiert werden verfügen NURBS-Objekte zusätzlich zum dreidimensionalen geometrischen Raum in dem sie angezeigt werden über einen Parameter-Raum. Insbesondere gibt eine Gruppe von Werten welche als Knoten bezeichnet werden den Einflussbereich eines jeden Steuer-Scheitelpunktes auf der Kurve oder Fläche an. Knoten sind im dreidimensionalen Raum unsichtbar, und lassen sich nicht direkt manipulieren, gelegentlich jedoch beeinflusst ihr Verhalten die Darstellungsweise des NURBS-Objektes. Der Parameter-Raum ist eindimensional für Kurven, die über eine einzige topologische Dimension (u) verfügen und zweidimensional für Flächen deren Dimensionen als u und v bezeichnet werden.

NURBS-Flächen haben im Wesentlichen die gleichen Eigenschaften wie NURBS-Kurven. Komplexe Flächen werden meist durch mehrere rechteckige Flächen-Segmente (Patches) modelliert, wobei aber an den Übergängen zwischen einzelnen Patches Kontinuitätsprobleme auftreten.

Siehe auch [4] und [7].

Um diese Kontinuitätsprobleme zu lösen wurden Gregory Patches entwickelt. Sie bestehen im Gegensatz zu den gängigen Patches nicht aus vier Kurven welche flach auf der zu modellierenden Fläche aufliegen (je zwei in u und in v Richtung), sondern aus fünf oder mehr Kurven welche außerhalb der Fläche liegen wodurch sich fliessendere Übergänge zwischen einzelnen Patches ergeben. Vergleiche hierzu Abbildungen 5 und 6.

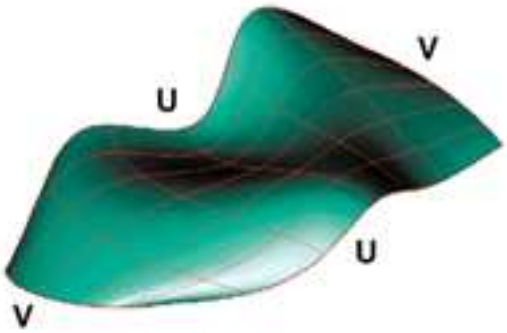

Abb. 5 - Coons-Patch bestehend aus zwei Kurven-paaren

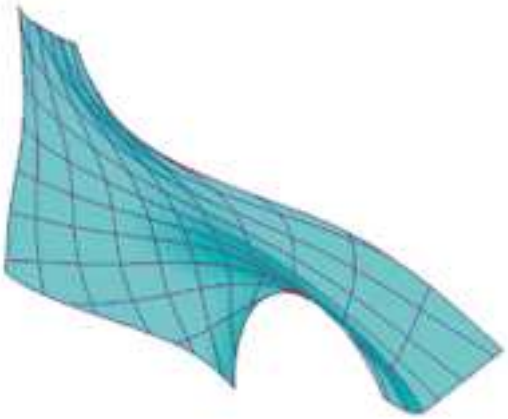

Abb. 6 - Gregory-Patch aus 6 Kurven welche alle nicht auf der Fläche aufliegen

Eine leicht verständliche Zusammenfassung findet sich unter [8]

2.4. Volumen-Modell

Das Volumen-Modell geht von der Verwendung voller Körper aus und ist somit die natürlichste Methode der Modell-Darstellung. Das Volumen der Objekte wird direkt in der Datenstruktur repräsentiert, so dass es für im Volumen-Modell erstellte Körper kein Problem ist, die richtige Sichtbarkeit zu ermitteln.

Mit dem Volumen-Modell können auch 3D-Objekte beschrieben werden, die keine explizite Oberfläche aufweisen. Dies können zum Beispiel Dichte-Felder, aber auch Wolken oder Feuer sein. Bei der Modellierung wird jeder Gegenstand entweder aus einfachen Elementar-Objekten wie Würfel, Kugel, Oktaeder, Pyramide, Kegel usw. aber auch aus komplexeren Elementar-Objekten wie Spur-Körper, Freiformkörper, Fraktale usw. zusammengesetzt. Beim Volumen-Modell handelt es sich um eine eindeutige Darstellungsform, bei der - im Gegensatz zu Draht- und Flächen-Modell - die Objekte immer

konsistent sind. Da die Objekte immer volle Körper sind, treten auch beim Schneiden keine Probleme auf.

Der Ansatz der Volumen-Grafik hat eine große Bedeutung in der Medizin und in nahezu allen Ingenieurs-technischen Disziplinen wie Maschinen- und Fahrzeugbau, Strömungstechnik, aber auch Bergbau oder Chemie. Beliebte Volumen-Modelle sind das Voxel- (von: Volume Element) Modell, Oktalbäume, Szenen-Graphen und Constructive Solid Geometry.

Chapter 3. Was bisher geschah...

Die Notwendigkeit Geometrie-Daten auszutauschen gibt es schon länger. Dabei kann man zwei große Themen-gebiete differenzieren: Konstruktionsdaten die mittels CAD (Computer Aided Design) für Fertigung und Simulation erstellt wurden und Daten für Echtzeit-Systeme. Diese Kategorien unterscheiden sich nicht nur in der Anwendungsart sondern auch in den Anforderungen. Während CAD-Systeme hohe Anforderungen an die Genauigkeit stellen liegen diese bei Echtzeit-Anwendungen eher bei Kompaktheit und ständig wachsenden Feature-wünschen von Anwendern und Entwicklern.

3.1. CAD

Beispiele für Formate aus dem CAD-Bereich.

Weitere Beschreibungen für CAD-Datenformate findet man zum Beispiel unter [2]

3.1.1. IGES

Der Initial Graphics Exchange Specification - kurz IGES - ist ein ANSI-kompatibles, nicht Hersteller-gebundenes Public Domain-Dateiformat, das als internationaler Standard zum Austausch von Produktdefinitionsdaten zwischen verschiedenen CAD/CAM-Systemen dient.

IGES wurde schon 1979 zum Datenaustausch zwischen CAD-Programmen und Vektor-basierten Anwendungen spezifiziert. Wie die meisten gängigen CAD-Formate kann IGES Geometrie auf verschieden Weisen speichern, von einfachen Linien und Kurven bis hin zu Freiformoberflächen wie Bezier- und NURBS-Flächen und Volumenelementen, letztere allerdings erst in der aktuellen 5.3 Version.

IGES spezifiziert sowohl ein Text- als auch ein Binärformat welches deutlich weniger Speicherplatz/Bandbreite benötigt, dabei aber die bekannten Vorteile von Text-Formaten aufgibt.

Aufgrund der relativ hohen Komplexität des Datenformats wird IGES fast ausschließlich von hochwertigen CAD-Applikationen genutzt. Trotzdem ist es ein wohldefiniertes und bewährtes Format.

3.1.2. CATIA

CATIA (Computer Aided Three-Dimensional Interactive Application) ist ein CAD/CAM Programm der Firma Dessault Systemes aus Frankreich. Diese entstand aus dem französischen Flugzeug-Hersteller Avions Marcel Dassault (heute Dassault Aviation) wo schon Anfang der siebziger Jahre mit der Entwicklung eines dreidimensionalen Grafikprogramms zur Unterstützung der Flugzeug-Ingenieure

begonnen wurde. Diese Applikation hat sich seither neben I-Deas, Unigraphics und ProEngeneer als Quasi-Standard im Fahrzeugbau etabliert, auch aufgrund der Tatsache dass CATIA Schnittstellen definiert mittels derer Dritt-Anbieter relativ einfach Zusatz-Funktionalität implementieren können.

CATIA verwendet verschiedene Dateiformate die allerdings - wie alle nicht standardisierten Formate - ständigen Änderungen mit neuen Versionen der Software unterliegen. Allerdings können gerade wegen dieser Flexibilität diese Datenformate alle Anforderungen der Nutzer erfüllen. .model oder .mdl Dateien sind das eigentliche Dateiformate von CATIA, zumindest bis einschließlich Version 4 der Anwendung. Seit Version 5 gibt es spezielle Formate zum Beispiel für Bauteile und -Gruppen. Daneben gibt es noch verschiedene sowohl Text als auch Binärexportformate.

CATIA wurde hier als Vertreter der nicht-offenen Standards ausgewählt da es stark - aber keineswegs ausschließlich - in der Automobil-Industrie verwendet wird und dort oft für sämtliche Prozessen vom computergestützten Entwurf bis zur Fertigung und sogar bis in den Verkauf eingesetzt wird.

3.2. Echtzeit-Formate

3.2.1. 3DS

Die Firma Autodesk hat mit 3D Studio ein weit verbreitetes Programm zur Erstellung von dreidimensionalen Modellen geschaffen welches sich im Hobby- sowie im Semi-Professionellen bis Professionellen Anwendungsbereich als äußerst erfolgreich erwiesen hat.

Das 3DS Format und seine Derivate sind zwar keine offenen en Datenaustausch-Formate, aber aufgrund der großen Popularität von 3DStudio hat es sich als de-facto Standard etabliert. Sie bieten ein äußerst breites Spektrum an Möglichkeiten und sind zumindest soweit offen als dass sich im Netz mehr als genug Anleitungen zum Einlesen dieser finden.

Da 3D Studio aber zu der Gruppe der Modellierungs- und Animationssoftware gehört bietet dieses Format auch verschiedene Möglichkeiten Animationen zu speichern.

Auch zu diesem Datenformat existiert ein Text-basiertes Pendant mit der Datei-Endung asc. Da die Frequenz der Spezifiaktionsänderungen das .3ds Format aber eher moderat ist hat sich hier in der Praxis die Binär-Variante durchgesetzt.

3.2.2. OBJ

Alias (früher Alias|Wavefront) ist Hersteller der 3D Modellierungs- und Animationssoftware Maya welche sich weitgehend als marktführend im High-End Bereich etabliert hat. Mit dem Text-basierten .obj

Format welches offen spezifiziert ist hat Alias außerdem ein auch unter anderen Anwendern sehr beliebtes Format geschaffen.

Das OBJ-Dateiformat unterstützt Linien, Polygone, Frei-Form-Kurven und -Oberflächen. Linien und Polygone werden mit Hilfe ihrer Punkte beschrieben, während Kurven und andere Oberflächen durch Kontrollpunkte und andere vom Typ der entsprechenden Kurve abhängigen Informationen beschrieben werden.

Eine weitere Variante des .obj Datei-Formates ist das Binärformat .mod, von welchem es aber so gut wie keine offene Dokumentation gibt und welches vom Leistungsumfang her mit .obj deckungsgleich sein sollte.

3.3. VRML

Die Virtual Reality Modeling Language (VRML) ist ein Mitte der Neunziger Jahre vom Web3D-Konsortium (damals noch VRML Architecture Group(VAG)) entwickelter Standard zur Speicherung und Darstellung von dreidimensionalen Strukturen. VRML wurde bewusst offen und mit dem Zweck der Darstellung zum Beispiel in Webbrowsern spezifiziert.

VRML basiert auf einem Szenen-Graphen in welchem die Knoten Objekte und Eigenschaften der darzustellenden Welt repräsentieren, sie enthalten statische Eigenschaften wie Geometrie und Farbe. Gruppen-knoten bieten die Möglichkeit Objekte zusammenzufassen, während Transformationsknoten dynamische Eigenschaften wie Translationen, Rotationen und Scheerungen beinhalten. Das Format bietet neben einfachen Geometrien und Materialien auch die Möglichkeit Verhaltensweisen zu definieren die zum Beispiel auf Annäherung der Kamera reagieren. Ein einfaches Beispiel:

```
#VRML V2.0 utf8
Transform {
    children [
        Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 0.0 0.0 0.9
                }
            }
            geometry Box {size 6.0 6.0 3.0}
        }
    ]
}
Transform {
    translation 10.0 0.0 0.0
    children [
        Shape {
            appearance Appearance {
                material Material {
```

```
                    diffuseColor 0.6 0.1 0.1
                }
            }
            geometry Sphere {radius 4}
        }
    ]
}
```

Mit der Zeit traten aber auch Nachteile zu Tage: Die starke Struktur des Formates sorgt zusammen mit dem Text-Format dafür dass die Dateien sehr groß werden und - was deutlich unangenehmer ist - dass der Inhalt der Datei erst dargestellt werden kann nachdem diese vollständig übertragen wurde. Auch merkt man dem VRML-Format das Alter an, so bietet es zum Beispiel nur unzureichende Möglichkeiten zur Darstellung komplexer Flächen, außerdem kann es den Anforderungen im sich schnell entwickelnden Echtzeit-Bereich nicht genügen, da es keine Methoden bietet moderne Features wie Shader oder spezielle Texturen für zum Beispiel Bump- oder Normalmapping zu repräsentieren.

Chapter 4. eXtensible 3D - X3D

Um die Nachteile von VRML97 aufzuräumen, hat das Web3D-Konsortium (http://web3d.org/) beschlossen einen Nachfolger zu spezifizieren. Da eine der Vorbedingungen war XML zur Formulierung der Beschreibungssprache zu verwenden entschied man sich dazu entschlossen nicht etwa eine neue Version von VRML sondern ein komplett eigenständiges Format zu entwickeln. Schließlich wurde im August 2001 die DTD eXtensible 3D (X3D) vorgestellt. Aktuell ist Version 3.0, die XSD-Spezifikation hierzu findet man unter http://www.web3d.org/specifications/x3d-3.0.xsd. Durch die Unterstützung von XML kann bei dreidimensionalen Szenen der Inhalt der Szene von der Darstellung getrennt und somit mehr Flexibilität erreicht werden. Prinzipien wie die Beschreibung einer Szene in Form eines hierarchischen Szene-Graphen oder die Methoden der Animation und Interaktion bleiben aus VRML erhalten. Im Unterschied zu VRML ist X3D jedoch modular aufgebaut (vgl. Abbildung 4).

X3D soll alle 3D-Standards auf einen Nenner bringen. Der Standard ist offen, Plattform-unabhängig und Lizenz-gebührenfrei. Spezielle Features können als Pakete eingebunden werden und müssen nicht von proprietären Programmen realisiert werden. Allerdings sind X3D-Dateien noch etwas größer als VRML-Dateien. Zudem ist der Standard noch relativ jung und wird erst durch wenige und z.T. nicht ausgereifte Programme genutzt.

Im Folgenden soll versucht werden die Grundzüge des X3D Formates anhand einiger Beispiele etwas genauer zu erörtern. Viele der Code-Beispiele hier stammen direkt aus oder wurden wenigstens inspiriert von [3]

4.1. Profile

Im Unterschied zur monolithischen Struktur von VRML97 - welche von der Anwendung eine vollständige Implementierung aller Features erfordert - erlaubt X3D Entwicklern nur bestimmte Untermengen der Spezifikation, sogenannte Profile, zu unterstützen, welche wiederum aus modularen Blöcken von Funktionalität - den Komponenten - zusammensetzen.

Diese Komponenten-basierte Architektur erlaubt aber nicht nur die separate Implementierung einzelner Profile, sondern ermöglicht auch die Neu-Einführung von Features in die Spezifikation ohne bereits existierende Implementierungen damit auszuschließen. Deswegen ist X3D ein sehr "lebendiges" Format welches ständig weiterentwickelt wird.

Die Basis-Profile von X3D sind:

- Interchange - dieses Profil beschreibt die Grund-Funktionalität zur Kommunikation zwischen Anwendungen. Es unterstützt Geometrie, Texturen, einfaches Lighting und Animation.
- Interactive - ermöglicht einfache Interaktion mit einer dreidimensionalen Szene indem dem Szenen-Graphen verschiedenste Sensor-knoten zur Navigation und Interaktion, fortgeschrittenes Timing und erweitertes Lighting hinzufügt.

- Immersive - definiert volle Features für 3D Graphik und Interaktion, inklusive Unterstützung für Audio, Kollisionsabfragen, Nebel und Scripting.
- Full - beinhaltet alle Knoten-Arten, inklusive NURBS-, H-Anim- und GeoSpatial-Komponenten.

Des weiteren ist es trivial einfach ein VRML97 Profil zu entwerfen, in welchem alle Eigenschaften von VRML97 enthalten sind und welches zusammen mit dem X3D Kern dessen volle Funktionalität gewährleistet. Damit ist X3D voll abwärts-kompatibel und es existieren einige Konverter, die VRML in X3D und umgekehrt umwandeln.

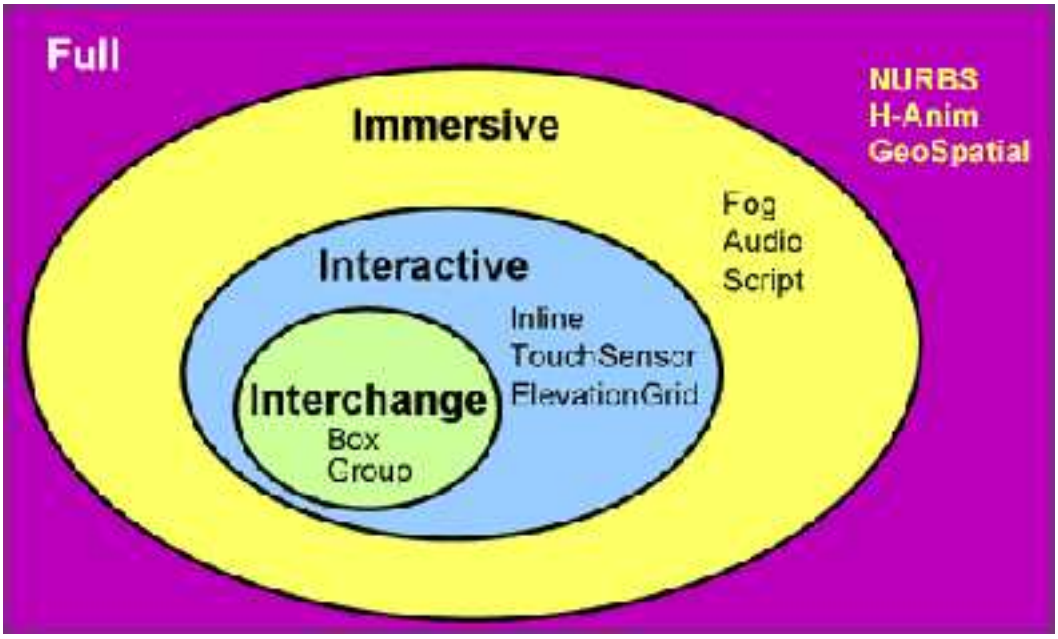

Abb. 4 - die X3D Baseline Profile

Welches Profil eine X3D Datei voraussetzt steht im <X3D> Knoten, welcher immer das Wurzel-Element einer X3D Datei ist. Dieser enthält außerdem noch übliche Namensraum-Angaben usw...

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"   "http://www.web3d.org/specifications

<X3D profile='Immersive'
     xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
     xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.0.xsd'>
</X3D>
```

4.2. Kopf, Szene und Navigation

Der Wurzel-knoten enthält in der Regel zwei Knoten, <head> und <Scene>. In <head> besteht die Möglichkeit einige Metainformationen wie zum Beispiel Autor und URL der Datei unterzubringen, während der <Scene>-Knoten die eigentlichen 3D-Daten enthält, zuvorderst den <NavigationInfo>-Knoten, welcher Größe, Position, Geschwindigkeit, Sichtweite, Typ usw. der Kamera definiert.

```
<head>
    <meta name="author" content="Michael Bayer"/>
    <meta name="created" content="11/17/2004"/>
</head>
<Scene>
    <NavigationInfo type="EXAMINE" "ANY"/>
</Scene>
```

4.3. Gruppen, Transformation, Geometrie und Material

Szenen setzen sich meistens aus <Shape>-Knoten zusammen, welche wiederum <Appearance>-Knoten zur Steuerung der Erscheinung (Material, Textur...) und die eigentliche Geometrie in Gestalt einfacher Formen (Box, Cone, Cylinder, Sphere...), aber auch Draht- und Flächen-Modelle aus Polygonen (Line, IndexedFaceSet...) oder komplizierteren Strukturen (NURBS, ElevationGrid...) enthalten. Des weiteren kann man jeglichen Unter-knoten einer Szene mittels <Group>-Knoten zu einer Gruppe zusammenfassen um eventuell einfach damit umgehen zu können, außerdem kann man um solche Unter-knoten auch einen <Transform>-Knoten stellen der statische Transformationen wie Rotationen, Translationen und Scheerungen enthält.

```
<Group DEF="IFCube">
    <Shape>
        <Appearance>
            <Material diffuseColor='0 0.5 1'/>
        </Appearance>
        <IndexedFaceSet coordIndex='0, 1, 2, 3, -1, 7, 6, 5, 4, -1, 0, 4, 5, 1, -1, 1, 5,
            <Coordinate point='-1.0 1.0 1.0, 1.0 1.0 1.0, 1.0 1.0 -1.0, -1.0 1.0 -1.0, -1
        </IndexedFaceSet>
    </Shape>
</Group>
```

Hier erkennt der geneigte Leser aber auch ein - der Meinung des Autors nach sehr nachteiliges - Feature von X3D: USE/DEF. Jeglicher Knoten-Typ innerhalb einer Szene kann mittels des Attributs "DEF=Name" einen Prototypen definieren der dann von anderen Knoten der selben Klasse per "USE=Name" wieder genutzt werden kann. Hier wurde anstatt eine der zahlreich vorhandenen Möglichkeiten Links in XML, wie zum Beispiel die XML Linking Language (XLL), zu nutzen eine eigene Methode entwickelt, weswegen leider der Entwickler beim Laden einer X3D-Datei nicht nur auf

bekannte Mechanismen zum Parsen von XML zurückgreifen kann sondern den erzeugten DOM-Baum nochmals "händisch" nach-bearbeiten muss.

4.4. Lighting

X3D bietet Unterstützung für die bekannten Beleuchtungsarten aus dem 3D-Bereich: ambientes Licht, direktionales Licht, Kugel- und Spotlights sowie Licht emmitierende Oberflächen. Die beiden ersten Licht-Arten werden aufgrund ihrer eher globalen Natur im <ViewPoint>-Knoten definiert, für die Restlichen Licht-Arten gibt es eigene Knoten. Eine besondere Licht-Art ist das sogenannte "HeadLight", eine Lichtquelle welche auf der virtuellen Kamera "montiert" ist und welche ebenfalls im <ViewPoint>-Knoten definiert oder besser abgeschalten wird, da es standardmäßig eingeschalten ist.

```
<NavigationInfo headlight='false' type='"EXAMINE" "ANY"'/>
<Group>
    <PointLight radius='12'/>
    <DirectionalLight ambientIntensity='1' direction='0 1 0' intensity='0.15'/>
</Group>
```

4.5. Sound

Ein weiteres interessante Feature das den Allgemeingültigkeitsanspruch dieses Formates unterstreicht ist die Unterstützung des Abspielens von Audio-Dateien. Neben ambienten Geräuschen welche keine bestimmte Richtung haben werden auch eigene <sound>-Knoten spezifiziert, welche genau wie alle anderen Knoten über ihre Position im Szenen-Graphen eine Richtung erhalten. Über diesen Knoten können sogar die Audiospuren von als Textur verwandten Filmen eine Position erhalten.

```
<Group>
    <Transform>
        <Sound minBack='5' minFront='5'>
            <AudioClip url=' "tone1.wav" "http://www.web3d.org/x3d/content/examples/Vrml2
        </Sound>
        <Sound maxBack='100' maxFront='100' minBack='30' minFront='30'>
            <MovieTexture USE='TV' containerField='source'/>
        </Sound>
    </Transform>
</Group>
```

4.6. Sensoren und Scripting

Wie auch schon VRML bietet X3D eine Reihe von Sensoren an welche zum Beispiel auf Annäherung, Sichtbarkeit, Kollision oder Berührung durch den Mouspointer reagieren und dadurch Aktionen wie Veränderung von Eigenschaften im Szenen-Graphen oder das Ausführen von eingebetteten Skripten ausführen können. Als Scripte können prinzipiell beliebige Scriptsprachen verwendet werden, allerdings muss das anzeigende Programm diese Sprache interpretieren können.

```
<Transform translation='-3 0 0'>
    <TouchSensor DEF='SensorButton1' description='Cyan background'/>
</Transform>
<Script DEF='Filter1'>
    <field name='set_boolean' type='SFBool' accessType='inputOnly' appinfo='filter input
    <field name='activated' type='SFBool' accessType='outputOnly'
        appinfo='activated provides a persistent binding value, otherwise isActive events
    <field name='count' type='SFInt32' value='1' accessType='initializeOnly'
        appinfo='count is only used locally, so it is declared as an interface for persis

    <![CDATA[


    ecmascript:

    function initialize () {
        Browser.print (' '); // skip line
        Browser.print ('Click shapes to select a background...');
        Browser.print (' '); // skip line
    }

    // setting the value of an eventOut variable also sends it as an event

    function set_boolean ( value, eventTime ) {
        // only trigger on true values so that Background stays bound
        if ( value == true ) {
            activated = value;
            Browser.print ('Cyan background ' + count);
        } else
        count++;  // received isActive value = false
    }

    ]]>
</Script>
```

4.7. Binary X3D

Da wie schon angesprochen X3D-Dateien dazu tendieren sehr groß zu werden wird im Moment an einer Spezifikation für ein "Binary X3D" gearbeitet, dieses ist im Moment allerdings noch im Vorschlagsstadium. Damit bei einem solchen Format nicht die Vorteile von XML zunichte gemacht werden wird aber angestrebt Binary X3D einfach als komprimiertes XML zu spezifizieren.

Chapter 5. XVL - eXtensible Virtual world description Language

Eine weitere Erweiterung zu VRML97 ist XVL und wird von der 1997 als Joint-Venture zwischen der Toyota Group und Matsua gegründeten Firma Lattice3D entwickelt. Das Verhältnis zwischen Lattice3D und Dessault wird nicht ganz klar, aber die Tatsache dass diese beiden Firmen gemeinsame Presse-Erklärungen abgeben lässt zumindest auf eine enge Zusammenarbeit schließen. XVL wurde unter etwas anderen Voraussetzungen entwickelt als X3D: Ziel war es dreidimensionale Konstruktionsdaten aus CAD-Anwendungen durch komplette Geschäfts-Prozesse hindurch zu nutzen, von der Konstruktion eines Produktes über die Fertigung bis hinein in den Verkauf. Hierzu ist es notwendig dass die Daten frei zwischen verschiedensten Applikationen ausgetauscht werden können, woraus schnell ersichtlich wird dass die Datenmengen die VRML bei genügender Genauigkeit produzieren würde zu groß wären. Lattice entwickelte nun eine Umgebung die diese Probleme lösen soll.

5.1. Lattice Structure

Der Lösung Ansatz den Lattice anstrebt ist nun der dass man Oberflächen nicht mehr durch Polygon-Netze anzunähern, sondern diese in eine Struktur umzuwandeln die Lattice "Lattice Structure" nennt. Diese besteht aus der Lattice Surface - einer Freiformoberfläche die durch sogenannte Gregory-Patches repräsentiert wird und dem Lattice Mesh welches die Kontrollpunkte der Gregory-Patches enthält.

Gregory-Patches sind eine besondere Art der parametrisierten Freiformoberflächen. Sie zeichnen sich durch eine sehr gute Oberflächen-Kontinuität aus, wodurch eine beliebige Oberfläche bei einer relativ hohen Genauigkeit durch sehr wenige Kontrollpunkte angenähert werden kann. Da Gregory-Patches verlustfrei aus NURBS erstellt und auch wieder in NURBS transformiert werden können können bekannte Algorithmen zur Darstellung genutzt werden. Siehe hierzu auch [4], Seiten 289-298 und [5].

Das Lattice Mesh ist ein Polygon-Netz welches aus den Kontrollpunkten für die Gregory-Patches sowie Attributen zur Wiederherstellung der eigentlichen Flächen besteht und die Selbe Topologie wie die darzustellende Lattice Surface hat. Dies sind unter anderem Angaben über den Grad der Rundung und das Ende derer.

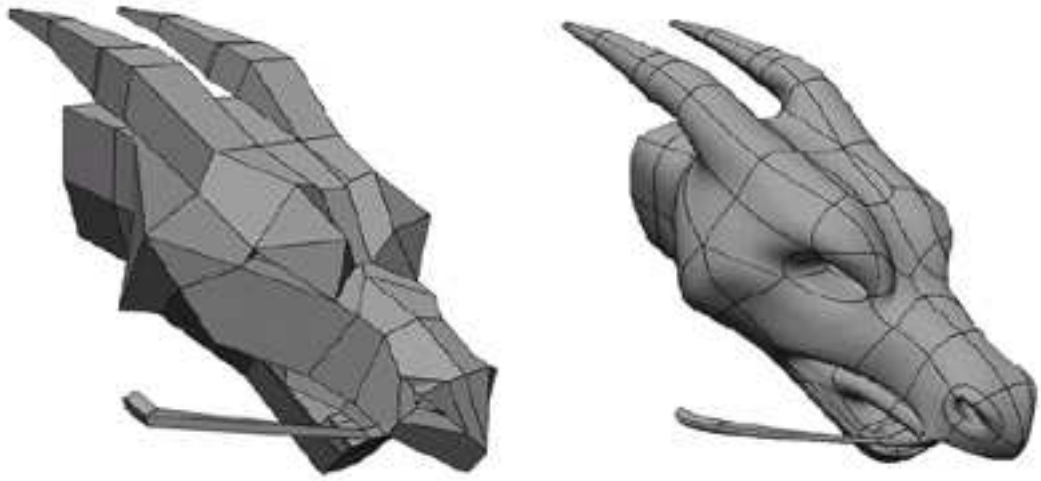

Abb. 1 - Objekt als Kontrollstruktur und Freiformoberfläche

5.2. Dateistruktur

Da das neue Format Internet-fähig sein sollte entschied man sich dafür eine Erweiterung zu VRML97 zu entwickeln. Da das Lattice Mesh nur aus Vertices und der Attribut-Struktur besteht kann man zur Speicherung dieser bereits vorhandene Strukturen des VRML-Formates nutzen. So wird das Lattice Mesh einfach als Polygon-Netz in bekannter VRML-Struktur gespeichert.

Die weiteren Informationen die nötig sind die ursprüngliche Fläche wiederherzustellen werden in einem VRML-Choice Knoten gespeichert welcher nur dargestellt wird wenn der darstellende Browser die XVL-Erweiterung beherrscht. Diese Methode hat zur Folge dass XVL-Dateien auch in normalen VRML-Browsern dargestellt werden können, wobei allerdings nur die Kontrollstruktur, nicht die eigentliche Oberfläche gerendert wird.

Weitere Änderungen am bestehenden VRML-Format werden nicht nötig, da XVL hauptsächlich auf den CAD/CAM-Bereich abzielt und dort die Genauigkeit der Repräsentation deutlich wichtiger ist als realistisches Aussehen der Modelle.

Ein denkbares Code-Beispiel, aus [1]:

```
#VRML V2.0 utf8

PROTO XVL_EDGE [
    field SFFloat round_val 0
    field SFVec3f round_str 0 0 0
    field SFVec3f round_end 0 0 0
    field MFInt32 is_round [1, 1, 1]
```

```
]
{
    Text{
        string["weight of edge rounding"]
    }
}
PROTO XVL_STATUS[
    field SFString status "XVL_LATTICE"
]
{
    Text{
        string["status of shape"]
    }
}
Group{
    children[
        ## Shape information of Lattice
        Group{
            children[
                Shape{
                    geometry IndexedFaceSet{
                    ############################
                    ## coordinate information ##
                    ## of Lattice Mesh ##
                    ############################
                    }
                }
            ]
        },
        ## Property information of Lattice
        Switch{
            choice[
                XVL_STATUS{
                    status "XVL_GREGORY"
                }
                XVL_EDGE{
                    round_val 0.5
                    round_str 0 1 1
                    round_end 0.2 0.3 1
                    is_round [1 1 1]
                }
                IndexedLineSet{
                    coordIndex[ 24 103 ]
                }
            ]
            whichChoice -1
        }
    ]
}
```

Der geneigte Leser wird nun feststellen dass dieses Code-Beispiel in keinster Weise irgendeiner XML-Spezifikation entspricht. Das liegt daran dass Lattice XVL zwar als "offenes XML-basiertes Format" anpreist, aber mit Beispiel-Dateien oder gar einer Offenlegung der Spezifikation aber sehr zurückhaltend - obiges Code-Beispiel ist das einzige das der Autor finden konnte, sonst waren nur Dateien zu finden die in der Binär-Version des Formates gespeichert wurden. Da sich solche Dateien nicht ohne weiteres öffnen lassen und Lattice auch hierzu keinerlei Information preisgibt war der Autor nicht in der Lage zu ergründen ob sich darin vielleicht XML verbirgt, aber selbst das würde ja sämtliche Vorteile von XML zunichte machen. Der Autor vermutet dass diese Zurückhaltung daher rührt, dass Lattice mit XVL eben nicht nur ein Format sondern ein komplettes Framework anbietet welches auch verkauft werden will.

5.3. Produkte

Wie kommt man also zu Dateien in diesem Format? Im CAD/CAM-Bereich wird schon jetzt sehr viel mehr als im Echtzeit-Bereich mit Freiformoberflächen gearbeitet, und da wie schon erörtert Gregory-Patches sehr einfach aus NURBS erstellt werden können gestaltet sich die Konvertierung solcher bestehender Daten als eher einfach. Komplizierter ist das Erstellen solcher Patches aus Polygon-Netzen. Zwar währe es sehr einfach das Polygon-Netz einfach als Kontrollstruktur zu verwenden, aber dabei ginge ja die zu speichernde Struktur verloren. Auch könnte man jedes Polygon relativ einfach mit einem einzelnen Patch beschreiben, aber hierbei würden wieder zu viele Daten entstehen. Um dieses Erstellungsproblem zu lösen bietet Lattice eine Reihe von Produkten und Services an:

Der XVL-Kernel ist Mittelpunkt jeder XVL-Applikation und ist als "Devolopment Toolkit" als Laufzeit-Bibliothek ausgeliefert. Er beherrscht das Laden und Speichern des XVL-Datei-Formates sowie die Konvertierung verschiedener Ausgangsstrukturen in die Lattice Structure.

Der XVL-Designer ist eine Freeware-Software die das Erstellen und Editieren einfacher Modelle unterstützt, allerdings nur mit sehr eingeschränkten Möglichkeiten. Das XVL-Studio hingegen ist eine komplette Modellingsoftware die zusätzlich den XVL-Composer - eine Betrachtungssoftware für CAD-Datenformate - und den XVL-Director - eine Animationssoftware für XVL-Modelle - integriert. Außerdem enthält es den XVL-Webmaster der aus XVL-Dateien Bauteil-Listen und -Bäume und 2D-Illustrationen erzeugt und diese in einer Web-Seite zusammenfasst.

Es existieren einige Browser-Plugins um diese XVL-fähig zu machen: den XVL-Viewer für Macintosh und den XVL-Player für Internet Explorer und den Netscape Navigator auf Windows-Plattformen. Von letzterem existiert auch eine Professional-Version welche Features für anspruchsvolle Benutzer wie zum Beispiel präzise Messungen bietet.

Der XVL-Signer ist eine Sicherheitsapplikation die es dem Anwender ermöglicht XVL-Dateien zu signieren und zu verschlüsseln. Außerdem bietet sie weitreichende DRM-Features, welche allerdings nur von zweifelhafter Nützlichkeit sind, da ein ambitionierter Nutzer sicher Mittel und Wege finden wird an die angezeigten Daten heranzukommen.

Weiterhin bietet Lattice eine Reihe von Daten-Konvertern an welche die Transformation verschiedenster gängiger CAD/CAM-Formate nach XVL und zurück unterstützen. Des Weiteren existiert ein XVL-Plugin für 3DStudio-Max.

Chapter 6. Fazit

Diese Ausarbeiten versuchte nachdem kurz erklärt wurde warum XML das Mittel der Wahl auf der Suche nach einem Datenaustauschformat ist einen kurzen Überblick über verschiedene Möglichkeiten zur Objekt-Repräsentation im 3D-Bereich von Punkt-Wolken über Drahtgitter- und Polygon-Netze bis hin zu Parameter-Oberflächen und Volumen-Modellen zu geben. In Kapitel 3 wurde kurz auf die beiden großen Bereiche in denen ein solches Format zum Einsatz kommen könnte - CAD/CAM und Echtzeitapplikationen - eingegangen und einige bisher verwendete und übliche Formate dieser beiden Bereiche vorgestellt. Danach wurden mit X3D und XVL zwei Kandidaten für ein solches neues Datenformat kurz vorgestellt und ihre jeweiligen Stärken und Schwächen aufgezeigt.

Die Vielzahl der Bereiche, in denen dreidimensionale Computergrafik heute eingesetzt wird zeigt dass Bedarf an einem einheitlichen Datenformat besteht. Die Anwendungsmöglichkeiten reichen von Computer-spielen und dreidimensionale Chatrooms über virtuelles Lernen, Produkt-Visualisierung, CAD und Industrie-Design, der Besichtigung virtueller Gebäude oder Städte bis hin zu methodischen Bereichen wie Medizin, Geologie und Stadtplanung. Insbesondere die wissenschaftlichen Disziplinen finden bei ihrer Suche nach Natur-naher und realistischer Verwaltung und Analyse oft in der dreidimensionalen Computergrafik eine Antwort.

Bezüglich der dreidimensionalen Dateiformate lässt sich sagen, dass sie stets unter Berücksichtigung ihres Einsatz-Zweckes untersucht werden müssen; ein Vergleich zwischen Formaten verschiedener Gruppen, z.B. zwischen einem CAD/CAM-Format und einem anwendungsspezifischen Format, ist in der Regel nicht sinnvoll.

Zu erkennen ist, dass sich bisher kaum einheitliche Standards durchgesetzt haben. In einem Großteil aller Fälle, in denen ein neues Grafik-Format entwickelt wird, treten keine neuen Funktionalitäten hinzu, so dass hier die Weiter-Verwendung eines vorhandenen Formats völlig ausreichend gewesen wäre.

Dies zeigt dass zumindest der Markt für ein allgemeines Datenformat besteht, ein solches steht oder fällt aber mit der Akzeptanz bei Anwendern und Entwicklern. Je mehr bestehende Applikationen ein Format unterstützen desto wahrscheinlicher ist es dass der Druck auf den Entwickler groß genug wird dieses auch zu implementieren - ein typisches Henne-Ei Problem für welches augenscheinlich keine Lösung in Sicht ist.

Die Vorteile eines solchen Formates liegen jedoch auf der Hand: Neben den Möglichkeiten zum Datenaustausch welche nur für den Anwender von Vorteil sind tritt auch immer mehr die Reduktion von Entwicklungszeit in den Vordergrund, da bestehende Bibliotheken genutzt werden können. Gerade letzteres lässt Hoffnung bestehen.

Die beiden vorgestellten Formate beanspruchen beide für sich Kandidaten für diesen ersten Platz unter den 3D-Formaten zu sein, wobei in der Literatur vor allem X3D eine weite Verbreitung vorhergesagt wird. Dieser Meinung muss sich der Autor anschließen, da XVL weder offen Spezifiziert noch ausreichend dokumentiert ist. X3D hingegen bietet nicht nur eine herausragende Dokumentation sondern

auch genug Features um alle Wünsche zufrieden zu stellen. Auch macht dieses Format den "lebendigeren" Eindruck und es fließen aus verschiedensten Quellen ständig neue Features in den Standard mit ein. Der einzige Vorteil den der Autor für XVL sieht ist das ausgereifte Framework von Applikationen die Lattice zu diesem Format anbietet. Den angepriesenen Vorteil der Daten-Kompression kann der Autor nicht in diesem Masse bewerten, da sowohl die Speicherkapazitäten von Festspeichern als auch die Bandbreiten in Kommunikationsnetzen immer größer werden.

Quellen-Verzeichnis

[1] Wakita, Yajima, Harada, Toriya, Chiyokura *XVL: A Compact And Qualified 3D Representation With Lattice Mesh and Surface for the Internet*. Keio University

[2] *http://www.datenaustausch.com/datfor.htm - CAD Datenformate*. datenaustausch.com - CAMTEX GmbH

[3] *VRML Sourcebook examples*. web3d.org

[4] Chiyokura, Kimura. *Design of Solids with Free-form Surfaces*.

[5] Kenjiro Takai Miuraa, Kuo-King Wangb C^2 *Gregory patch* http://www.eg.org/EG/DL/Conf/EG91/papers/EUROGRAPHICS_91pp481_492_abstract.pdf.

[6] Ling Huang JianFeng Zhen Xinxiong Zhu Leiyi. *A Surface Interpolating Method for 3D Curves-Nets* http://www.cs.berkeley.edu/~hling/research/paper/surface.htm

[7] C.-K. Shene. *Introduction to Computing with Geometry Notes* http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/

[8] Joe Greco. *Surfaces*. http://www.deskeng.com/articles/00/Dec/feature_MCAD/main.htm

[9] web3D Consortium *Open Standards for Real-Time 3D Communication* http://web3d.org/

Appendix A. Abbildungsverzeichnis