1 Motivation

Die Anfänge von VRML liegen nun bereits einige Jahre zurück, der Standard wurde erweitert und verbessert. Eine Vielzahl von Playern und Autorenwerkzeugen wurden entwickelt und ermöglichen es heute den Einsatz von VRML auch schon für umfangreichere, möglicherweise kommerzielle Produktionen zu planen und zu realisieren.

Dies heißt aber auch, daß Welten zunehmend nicht mehr relativ einfache, handgestrickte Gebilde mit demonstrationscharakter sind, sondern komplexe, mit der Unterstützung ausgefeilter Werkzeuge erstellte Projekte. Nicht zuletzt auch durch Entwicklungen wie „living worlds“ wächst die Komplexität und Größe der VRML Dateien.

Auf der anderen Seite sind die Übertragungsleistungen im Internet, über das ja die meisten dieser Projekte verbreitet werden, noch immer beschränkt. Wenn zudem als Zielgruppe der zukünftig „normale Durchschnittskonsument“ zu Hause mit seiner typischerweise extrem schmalbandigen Internetanbindung avisiert ist, so stellen Übertragungszeit und übertragenen Datenmenge entscheidende Kostenfaktoren dar. Zusätzlich benötigen die Projekte Speicherplatz auf Datei-Servern und Anwendungsrechnern. Nicht zu vernachlässigen ist auch der durch die gestiegene Komplexität der Dateien wachsende Aufwand eine solche Datei zu parsen.

Eine naheliegende Lösung wäre eine transparente Kompression / Dekompression mit vorhandenen Packern wie *gzip* oder ähnlichem. Allerdings ist die Kompressionsrate vergleichsweise bescheiden und das Parsen der Datei würde in keiner Weise beschleunigt.

Einen umfassenderen Ansatz strebt die „*VRML Compressed Binary Format working group*“ in [3] an.

Im Mittelpunkt dieser Arbeit stehen die von der Gruppe entworfenen Mindestanforderungen an einen entsprechenden Standard, erste Enwürfe eines Standards, die darin verwendeten Techniken und die erzielbaren Ergebnisse.

2 Erste Schritte

2.1 Wie alles begann

Die Notwendigkeit eines Binärformates für VRML wurde schon früh erkannt. Eigentlich sollte es Bestandteil der VRML 2.0 Spezifikation sein, aber um einen Konsenz unter den Mitgliedern des VRML Konsortiums über die Spezifikation nicht zu gefährden hat man die Standardisierung des Binärformates verschoben.

Der erste Vorschlag („Request for proposal“) für einen Binärformatstandard wurde 1996 veröffentlicht.

In der Folgezeit wurde intensiv an den Problemen gearbeitet, aber es war schwierig eine gemeinsame Basis zu finden. Da man durch diese Verzögerungen viel Zeit verloren hatte und die Entwicklungen in Technologie und Wirtschaft nicht ignorieren wollte, wurden die Anforderungen an den Standard Anfang dieses Jahres überarbeitet. Einige Auszüge aus diesem Dokument werden im folgenden kurz dargestellt.

2.2 Aktuelle Anforderungen an den Standard

Plattformunaghängigkeit	Die Algorithmen sollen in einer auf verschiedenen Plattformen verbreiteten Programmiersprache umsetzbar sein und nicht von bestimmter Hardware oder einem bestimmten Betriebssystem abhängig sein.
Offenheit	wie VRML allgemein, so muß auch dieser Standard auf Technologien basieren, die offen und ohne Lizenzgebühren verfügbar sind
Leistungsfähigkeit	Geschwindigkeit ist ein wichtiger Fakor beim Einsatz von VRML. Der Kompressionsfaktor hat einen hohen Stellenwert, da er die Ladezeiten erheblich verkürzt. Zusätzlich sollten aber auch die Dekompressionszeiten optimiert werden.

Tabelle 2-1. Die wichtigsten Anforderungen an den Standard

Kompression	Ein minimaler Faktor wäre 1:5, dieser ist bereits durch herkömmliche Kompressionsverfahren sowie Optimierung des UTF Codes erreichbar.
Qualität	Die Kompression muß, wenn nicht vom Autor der Welt explizit anders gewünscht, verlustlos sein. Das heißt, das eine gepackte Welt genauso aussehen muß wie die entsprechende unkomprimierte. Außerdem müssen Kompression und Dekompression mehrfach hintereinander ausführbar sein, ohne das ein Datenverlust eintritt.

Tabelle 2-1. Die wichtigsten Anforderungen an den Standard

Die komplette aktuelle Anforderungsliste der *„VRML Compressed Binary Format Working Group“ (CBFWG)* ist unter [4] nachzulesen.

3 Die Umsetzung

Einer der bedeutendsten Standardentwürfe im Bereich CBF (Compressed Binary Format) entstammt einer Zusammenarbeit von *IBM* und *ParaGraph international Inc. [1]*. Dieser Entwurf bildet die Grundlage der weiteren Betrachtungen. Es werden die verwendeten Techniken, deren theoretische Hintergründe und die bereits erreichten Ergebisse vorgestellt.

Die verwendete Basistechnologie, ist die *Geometrische Kompression durch topologische Operationen*, eine Arbeit von G. Taubin und J. Rossignac, entstanden 1996 innerhalb der *Visual and Geometric Group at IBM Research* [6]. Diese Basistechnologie wurde um die speziellen Anforderungen von VRML2.0 erweitert und liefert Kompressionsraten bis zu 50:1. Der Algorithmus komprimiert zusammenhängende geometrische Daten auf bis zu zwei Bit pro Dreieck, natürlich verlustlos. Kodierungen, die auf Wunsch auch verlustbehaftet arbeiten, werden für die Darstellung von räumlichen Koordinaten, Farben, und Texturkoordinaten verwendet.

3.1 Wie funktioniert das ?

Im Grunde genommen ist das Binärformat eine Umschreibung des zugrundeliegenden ASCII Formates mit einigen Erweiterungen, die nötig sind, um die topologiebasierte Kompression von verschiedenen Knoten zu ermöglichen. Das bedeutet, daß jeder Knoten in der ASCII Datei einen assoziierten Knoten in der Binärdatei besitzt. Insbesondere wird der Szenengraph an sich bei dieser Umwandlung nicht verändert. Die verschiedenen Namen der Knotentypen erhalten dazu einfach ein äquivalentes Binärtoken. Zusätzlich zur direkten Übersetzung von ASCII in das Binärformat stehen zwei Kompressionsmechanismen zur Verfügung. Der eine, als *field compression* bezeichnet, ermöglicht die Komprimierung von folgenden VRML Feldtypen: MFColor, MFFloat, MFInt32, MFRotaion, MFTime, MFVes2F und MFVec3F.

Der zweite Mechanismus, als topologiebasierte Kompression bezeichnet, erlaubt die Verarbeitung der Knotentypen CoordinateInterpolator, ElevationGrid, IndexedFaceSet, PointSet sowie NormalInterpolator.

Die topologiebasierte Kompression kodiert dabei sowohl die topologischen Daten, als auch die Eigenschaftseinträge der Knoten. Die Kompression von Topologiedaten ist dabei verlustlos, andere Daten können wahlweise auch verlustbehaftet komprimiert werden.

3.1.1 Wie funktioniert die topologiebasierte Kompression ?

Die geometrischen Daten in den VRML Knoten stellen meist Dreiecksnetze dar, oder können leicht in solche überführt werden. Dreiecksnetze lassen sich gut als aus Dreiecken aufgebautes Drahtgittermodell eines Objektes vorstellen. Ein guter Ansatz zur Kompression topologischer Daten ist daher die effiziente Darstellung dieser Netze. Ein Dreiecksnetz ist bestimmt durch die geometrische Information über die Lage der Dreieckseckpunkte, durch die Verbindungsinformationen, die die Zugehörigkeit der Eckpunkte zu den Dreiekken bestimmt, und durch Farb- und Texturinformationen, die das Aussehen der Objekte und die Oberflächeneigenschaften bestimmen. Die meisten Kompressionsalgorithmen basieren daher auf der effizienten Darstellung dieser drei Informationen.

Die hier betrachtete, in [1] ausführlich erklärte, Kompression stellt jedes Dreiecksnetz durch zwei Bäume, den Dreiecksbaum (triangular mesh) und den Verbindungsbaum (vertex mesh) dar. Diese Bäume werden in den speziellen VRML Knoten *_SFBinary VertexForest*, *_SFBinary TriangleForest*, und *_SFBinary DataRecord* abgelegt.

Der Verbindungsbaum spannt einen Graphen bestimmt durch die Kanten und Eckpunkte des Dreiecksnetzes auf. Dabei ist jede Kante genau einmal im Graphen enthalten.

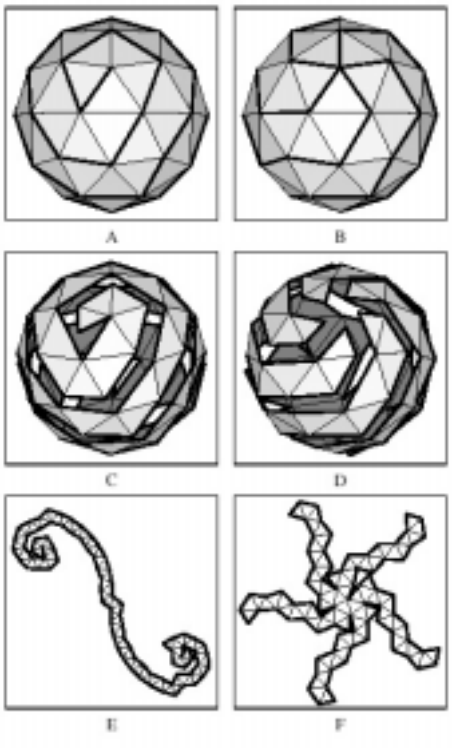

Abb. 3-1. A,B: beispielhafte Verbindungsbäume auf einem Dreiecksnetz; C,D: aufgeschnittenes Dreiecksnetz; E,F: mögliche Dreiecksgraphen des Netzes

Zu Beginn des Kompressionsprozesses wird das Dreiecksnetz entlang der Kanten des Verbindungsbaumes aufgeschnitten. Die Kanten des Verbindungsbaumes werden daher auch als Schnittkanten bezeichnet. Der zu einem aufgeschnittenen Dreiecksnetz gehörende Graph wird Dreiecksgraph genannt. Ein Eckpunkt im Dreiecksgraphen gehört dabei zu einem Dreieck im aufgeschnittenen Dreicksnetz und eine Kante im Dreiecksgraphen ist eine zu zwei aneinandergrenzenden Dreiecken des Dreiecksgraphen gehörende Kante. Aus einem Dreiecksgraph läßt sich dann der Dreiecksbaum generieren. Der beschriebene Vorgang ist in Abb. 3-1 auf Seite 6 grafisch dargestellt.

Jeder Verbindungsbaum ist eine Sequenz von Verbindungsläufen (runs). Ein Verbindungslauf ist ein Pfad des Baumes, welcher einen Blatt- oder Verzweigungsknoten mit einem anderen Blatt- oder Verzweigungsknoten verbindet, ohne einen weiteren Verzweigungsknoten zu kreuzen. In Abb. 3-2 auf Seite 6 sind die einzelnen Läufe in verschiedenen Farben dargestellt. Das zugrundliegende Dreiecksnetz und der Dreiecksgraph sind ebenfalls in Abb. 3-2 zu sehen. Ein als Wurzelknoten ausgewählter Blattknoten bestimmt für jeden Lauf

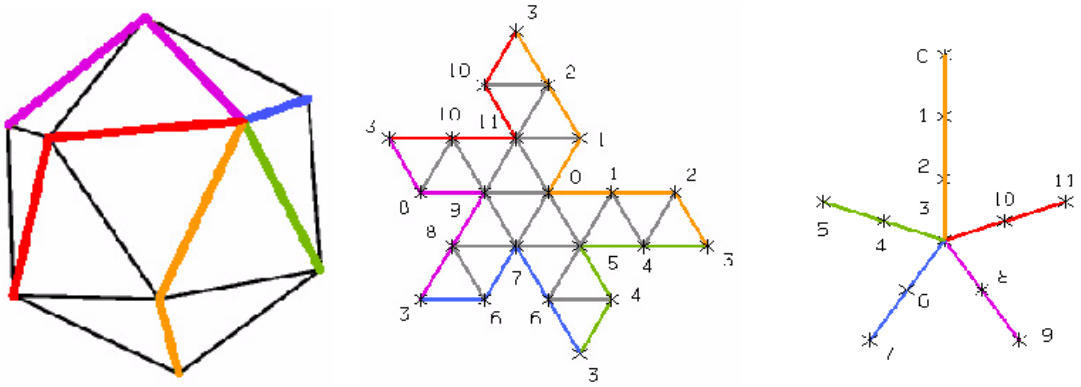

Abb. 3-2. v.l.n.r. Dreiecksnetz mit Verbindungsbaum; Dreiecksgraph; Verbindungsgraph mit den einzelnen Läufen

Anfangs- und Endknoten. Man betrachtet die Läufe von einem Verzweigungsknoten ausgehend entgegen dem Uhrzeigersinn. In der sich daraus ergebenden Reihenfolge werden die Läufe auch im entsprechenden Knoten der Datei abgelegt. Jeder Verbindungslauf wird mit einem Ende-Bit, einer Längenangabe und einem Blatt-Bit abgelegt. Das Ende-Bit gibt dabei an, ob der Lauf der letzte im Verbindungsbaum ist, die Längenangabe enthält die Anzahl der Kanten im Lauf, und das Blatt-Bit zeigt an, ob der letzte Knoten des Laufes ein Blatt- oder Verzweigungsknoten ist.

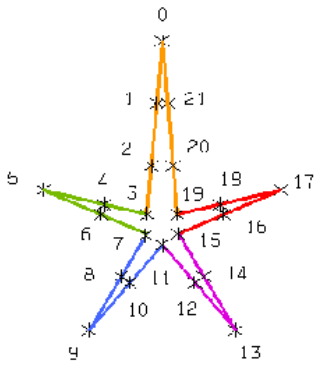

Abb. 3-3. Die Randschleife des Verbindungsgraphen.

Der Rand des Dreiecksgraphen wird Randschleife genannt. Jede Kante des Verbindungsbaumes gehört zu zwei Kanten der Randschleife. Diese Zusammengehörigkeit bestimmt, welche Kanten bei der Dekompression „zusammengenäht“ werden müssen, um das originale Dreiecksnetz zu rekonstruieren.

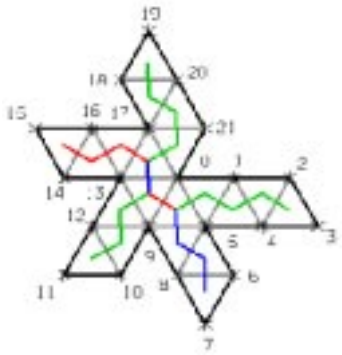

Abb. 3-4. Der Dreiecksgraph mit eingefärbten Dreiecksläufen.

Ein Dreiecksgraph muß nicht immer auch ein Dreiecksbaum sein, da er zyklisch sein könnte, er kann aber durch das entfernen von Kanten in einen solchen überführt werden. Dabei entstehen sogenannte Sprungkanten, die einen Randschleifenknoten mit einem anderen verbinden. Ein Dreieck des Dreiecksbaumes welches von drei Dreiecken umgeben ist, heißt Verzweigungsdreieck. Ein Dreieckslauf ist ein Pfad im Dreiecksbaum, welcher ein Blatt- oder Verzweigungsdreieck mit einem anderen Blatt- oder Verzweigungsdreieck verbindet, ohne ein Verzweigungsdreieck zu kreuzen.

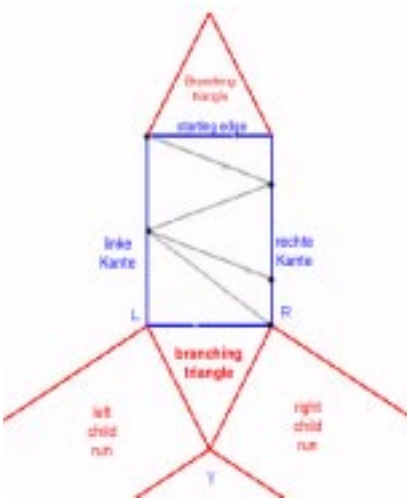

Abb. 3-5. Darstellung der Dreiecke eines Laufes mittels marching bit pattern

Die Dreiecke eínes Laufes bilden einen Streifen. Man legt einen der Blattknoten als Wurzel fest und kann dann anhand der Orientierung für jedes Verzweigungsdreieck einen linken und rechten (Kind-) Lauf benennen. Die Darstellung des Dreiecksbaumes in den Knoten der Binärdatei ähnelt stark der des Verbindungsbaumes. Zusätzlich wird für jeden Dreieckslauf die Information über die Eigenschaften der Dreiecke innerhalb des Laufes gespeichert. Dies geschieht in des sogenannten „Marching bit patterns“. Da die zu beschreibenden Läufe Streifen sind, deren linker und rechter Rand durch Kanten der Randschleife beschrieben sind und das Wurzeldreieck einen definierten Anfang bildet, lassen sich die weiteren Dreiecke durch nur ein Bit darstellen, welches beschreibt, ob der nächste Eckpunkt im linken, oder rechten Rand enthalten ist. Grafisch dargestellt ist der Sachverhalt in Abb. 3-5.

3.1.2 Wie funktioniert die Feldkoprimierung ?

Die Komprimierung der Knotentypen MFColor, MFFloat, MFInt32, MFRotaion, MFTime, MFVes2F und MFVec3F erfolgt in konventioneller Weise mittels Lauflängenkodierung um Redundanzen in Bitfolgen zu kodieren, sowie mittels Huffmankodierung um Redundanzen durch Häufigkeitsunterschiede von Bitfolgen zu vermeiden.

Daneben steht auch eine verlustbehaftete Komprimierung zur Verfügung, welche erwartungsgemäß bessere Kompressionsraten ermöglicht, aber nicht die Originalszene wiederzugeben vermag.

4 Ergebnisse

Zu dem dieser Arbeit zugrundeliegenden Standardisierungsvorschlag ist mir leider keine Implementierung bekannt. Die entsprechenden Seiten der IBM im Internet[5] bieten jedoch eine vorläufige Implementierung der 4.Version des Standardisierungsvorschlages[2]. Eine verfügbare Implementierung für Windows95 ist Grundlage der im folgenden vorgestellten Ergebnisse. Die verwendeten Grafiken stammen von den Seiten der IBM, die dargestellten Ergebnisse wurden aber an eigenen Beispielen aus verschiedenen Quellen auf Erreichbarkeit und Plausibilität geprüft. Außerdem ist die Informationsbasis mit einer Menge von 650 verschiedenen Szenen aus verschiedensten Quellen recht breit.

Die Implementierung besteht im wesentlichen aus drei Teilen. Das wichtigste Programm CVBF_Bin dient zur Umwandlung ASCII -> Binär und zurück. Neben der reinen Umsetzung erfolgt natürlich auch die in “Wie funktioniert die topologiebasierte Kompression ?” auf Seite 5 beschriebene Kompression. Allerdings werden bis jetzt erst wenige Knotentypen tatsächlich komprimiert. CoordinateInterpolator, ElevationGrid, IndexedFaceSet und NormalInterpolator sind nur einige Beispiele für Knoten die deren Kompression noch nicht immer optimal durchgeführt wird.

Das zweite Programm, CVBF_opt genannt, dient der Aufbereitung der topologischen Daten. Diese ist nötig, da die topologiebasierte Kompression bei bestimmten topologischen Eigenheiten sonst nicht anwendbar ist.

Als letzte Komponente wurde ein PlugIn für Webbrowser entwickelt, welches die Verwendung von VRML Binärdateien auch mit den vorhandenen VRML Viewern ermöglicht. Da jedoch dabei einfach eine temporäre VRML ASCII Datei erzeugt und dem Viewer zur Verfügung gestellt wird geht die mögliche Verkürzung der für das Parsen der Datei notwendigen Zeit vollständig verloren.

Nach all diesen Einschränkungen nun aber zu den ersten Kompressionsergebnissen, die für sich sprechen.

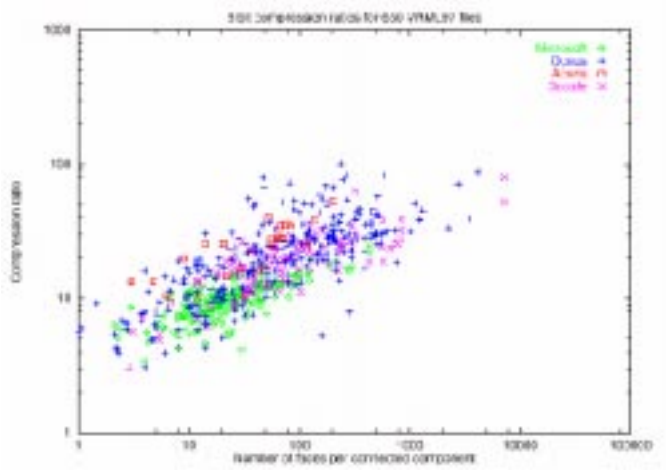

Abb. 4-1. Die Kompressionsraten in Abhängigkeit von der Komplexität der Objekte

Neben den Einsparungen durch die reine Kodierung / Kompression ist sicher auch sehr interessant welchen Vorteil man gegenüber herkömmlichen Verfahren wie dem sehr verbreiteten gzip Algorithmus erzielt.

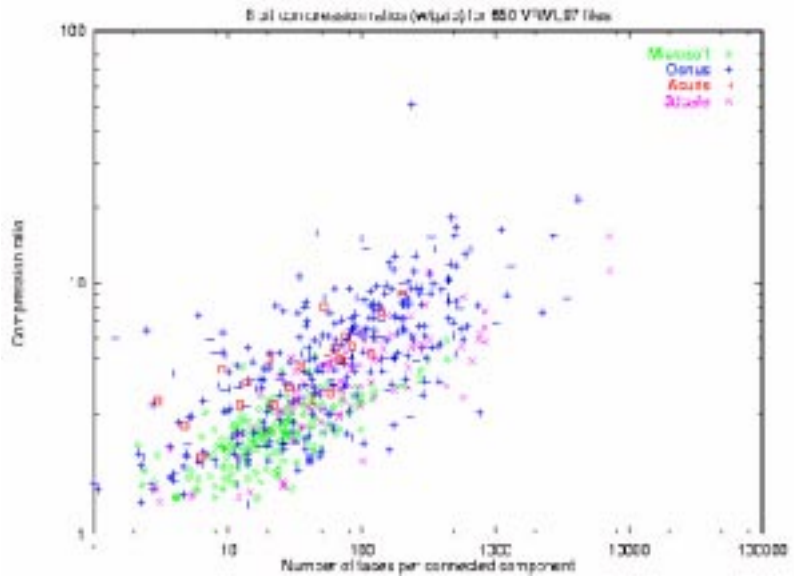

Abb. 4-2. Die Kopressionsraten von VRMLB im Vergleich zu gezippten VRML ASCII Dateien

5 Ausblick - Was bleibt zu tun?

Nachdem auf den letzten Seiten ausführlich die Ziele, Anforderungen und gegenwärtigen Ergebnisse dargestellt wurden sollen die folgenden Zeilen einen Ausblick auf das geben was uns noch erwarten könnte.

Der wichtigste Schritt scheint mir zunächst die Verabschiedung eines verbindlichen Standards, vielleicht basierend auf dem hier vorgestellten Vorschlag. Dieser Standard sollte von möglichst vielen Firmen unterstützt werden, um eine gewisse Durchsetzungskraft zu erreichen. Da die Größe der VRML Dateien für professionelle Projekte mit breitem Publikum ein wichtiger Faktor ist sollte man sich schnell einigen, um die Entwicklung solcher Projekte und damit die Verbreitung von VRML im allgemeinen nicht zu behindern.

Die bisherige Umsetzung des Standardisierungsvorschlages sollte in Zusammmenarbeit mit der VRML Compressed Binary Format Working Group weiterentwickelt werden, um in einer Musterimplementierung die vollen Möglichkeiten der topologiebasierten Kompression zu demonstrieren. Dies wäre eine gute Basis um die beeindruckenden Leistungen des verwendeten topologieorientierten Ansatzes mit einer großen Verbreitung und Anwendungsvielfalt zu verbinden.