

Optimierte Anzeige von Lego-Modellen

Ein OpenGL-Viewer für LDraw Modelle

von

Felix Funke

Technische Universität Braunschweig

Institut für Computergrafik

Prof. Fellner

Betreuung und Aufgabenstellung: S. Havemann
Braunschweig, 25. April 2003

Inhaltsverzeichnis

1	Einleitung	1
2	LDraw	2
2.1	Die Geschichte von LDraw	2
2.2	Das DAT Format	3
2.3	Probleme mit der Legobibliothek	5
2.3.1	Orientierung der Flächen	6
2.3.2	Schlechte Modellierung	7
2.3.3	Bowties	8
3	Der LegoEditor	10
3.1	Optimierungen	10
3.1.1	Two Sided Lighting	10
3.1.2	Vertex Arrays	11
3.1.3	Entfernen der Rekursion	11
3.2	Die Lego API	11
3.2.1	LegoObject	11
3.2.2	LegoModelData	12
3.2.3	LegoScene	12
3.3	Der LegoEditor	13
3.4	Weitere Optimierungen	14

1 Einleitung

Die Anzeige von Lego-Modellen wurde bereits von Tomas Neumann in seiner Studienarbeit behandelt. Das Ziel dieser Studienarbeit sollte sein die Darstellung so weit zu beschleunigen, dass es möglich ist interaktiv mit den einzelnen Legosteinen zu arbeiten.

Es gib im Internet eine große Gemeinschaft von Lego-Enthusiasten, die die meisten real existierenden Legoteile als 3D Modelle erstellt haben. Zur Anzeige dieser Modelle diente früher einmal LDraw. Inzwischen gibt es jedoch komfortablere Tools. Was geblieben ist, ist das Dateiformat und Inkonsistenzen. Im Laufe der Studienarbeit wurden einige Fehler, die beim Einlesen solcher Dateien auftreten können, erkannt und Möglichkeiten eronnen, diese Fehler zu beheben, so dass eine fehlerfreie Darstellung möglich ist.

Das Ergebnis der Studienarbeit ist eine kleine Bibliothek von Objekten, die es einem Programmierer ermöglichen Lego-Modelle in sein Projekt zu importieren und mittels OpenGL anzuzeigen. Ein kleiner Editor für Lego-Modelle dient als Anwendungsbeispiel und Demonstration der Leistungsfähigkeit der Bibliothek.

2 LDraw

2.1 Die Geschichte von LDraw

1995 wurde LDraw von James Jessiman entwickelt. LDraw diente zur Anzeige von Lego-Modellen. Mit einem speziellen Editor - LEdit - konnte man diese Modelle bearbeiten. Beides waren Programme für MS-DOS. Mit LDraw war es möglich alle Arten von Legomodellen im Rechner nachzubauen und anzuzeigen. Dazu mußten jedoch alle Legoteile, die es real gibt modelliert werden. Diese Aufgabe nahm sich eine Gruppe Begeisterter an und inzwischen ist fast jedes Legoteil in der Legobibliothek vorhanden.

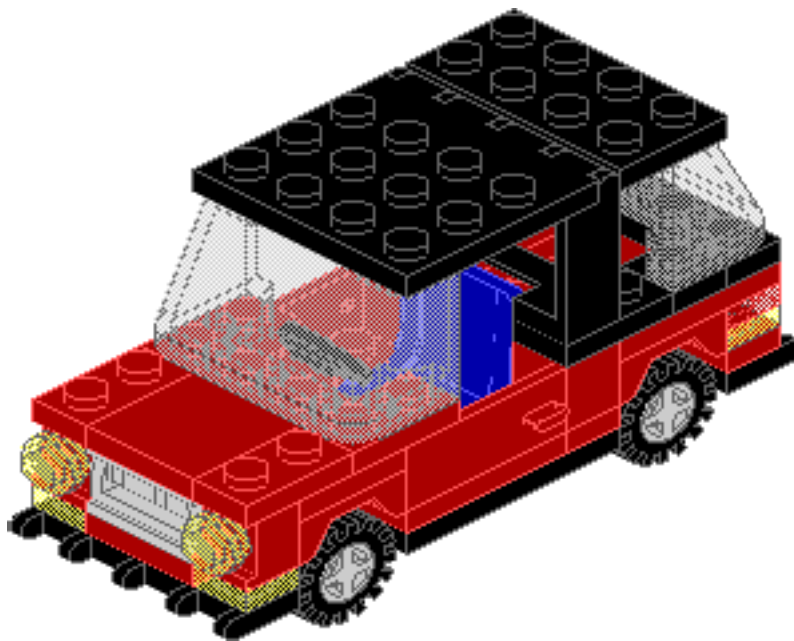


Abbildung 1: car.dat mit LDraw gerendert

Inzwischen gibt es natürlich auch weitere Programme zur Anzeige und zur Bearbeitung von Legomodellen. So ist die Ausgabe von LDraw auf die üblichen VGA-Grafikmodi beschränkt. Um jedoch hochauflösende komplett beleuchtete Bilder zu erzeugen wurde ein Konverter entwickelt, der ein Legomodell in ein POV-Ray Modell umwandelt (siehe Abbildung 2).

Natürlich sind solche Berechnungen äußerst aufwendig und für eine Echtzeitanzeige ungeeignet. Inzwischen gibt es eine Vielzahl an Programmen und kleinen Tools für verschiedenste Betriebssysteme um Legomodelle anzuzeigen. Einige nutzen inzwischen auch 3D beschleunigte Hardware aus. Allerdings haben diese Programme mit

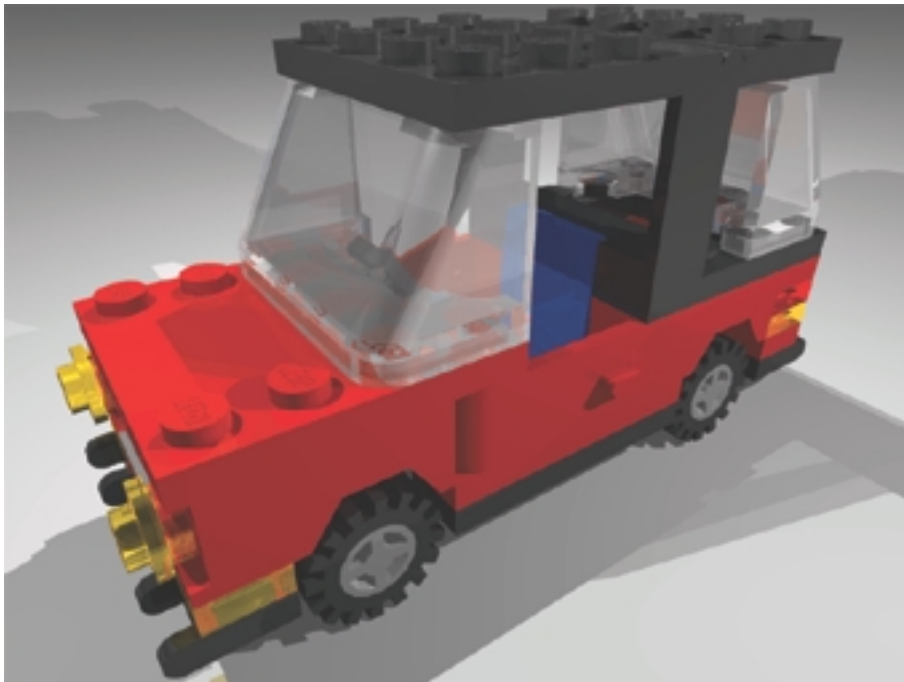


Abbildung 2: car.dat mit POV-Ray gerendert

den Altlasten der Entwicklung des Projekts zu kömpfen, wie weiter unten beschrieben wird.

2.2 Das DAT Format

Das DAT Format für Lego-Modelle ist eine Sammlung aus Textdateien, die die Geometrieinformationen einzelner Legoteile beinhalten. In jeder dieser Dateien kann es Verweise auf andere Dateien geben. So wird ein Legomodell Stück für Stück hierarchisch aufgebaut. Die Namensgebung der Dateien ist auf den ersten Blick jedoch recht verwirrend und unübersichtlich. Zum einem sind die Dateien der Lego-Bibliothek auf 8+3 Zeichen beschränkt. Außerdem wurden alle Dateien, die einzelne Legoteile repräsentieren nach der Modellnummer des entsprechenden realen Legoteils benannt. So heißt z.B. ein 2x4 großer Block 3001.dat.

Der Aufbau einer einzelnen Datei ist recht simpel. In jeder Zeile ist ein Kommando für den entsprechenden Parser abgelegt. Das Kommando ist das erste Zeichen in dieser Zeile. Den Rest der Zeile machen die Parameter des Kommandos aus. Die Bedeutungen der einzelnen Kommandos sind in Tabelle 1 aufgezeigt. Hier der Inhalt der Datei 3001.dat, der das Modell in Abbildung 3 erzeugt.

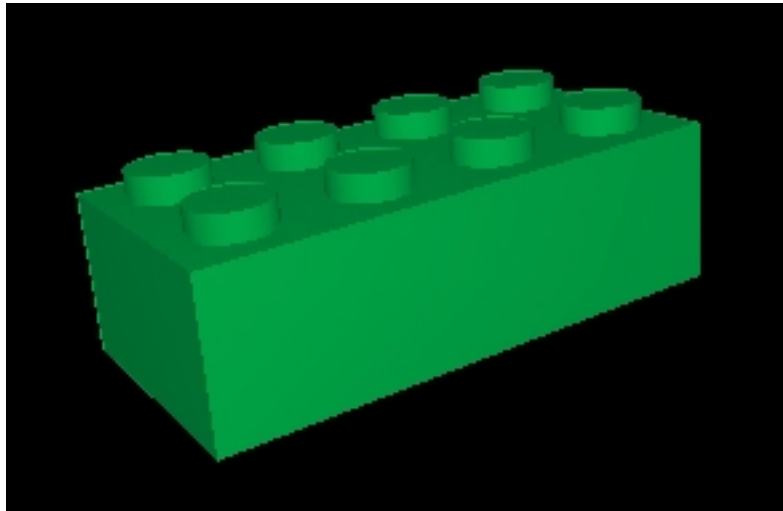


Abbildung 3: 2x4 Block - 3001.dat

```

0 Brick 2 x 4
0 Name: 3001.dat
0 Author: James Jessiman
0 Original LDraw Part
0 LDRAW_ORG Part UPDATE 2002-03
0 BFC CERTIFY CCW
0 2002-05-07 KJM BFC Certification
1 16 20 4 0 1 0 0 0 -5 0 0 0 1 stud4.dat
1 16 0 4 0 1 0 0 0 -5 0 0 0 1 stud4.dat
1 16 -20 4 0 1 0 0 0 -5 0 0 0 1 stud4.dat
0 BFC INVERTNEXT
1 16 0 24 0 36 0 0 0 -20 0 0 0 16 box5.dat
4 16 40 24 20 36 24 16 -36 24 16 -40 24 20
4 16 -40 24 20 -36 24 16 -36 24 -16 -40 24 -20
4 16 -40 24 -20 -36 24 -16 36 24 -16 40 24 -20
4 16 40 24 -20 36 24 -16 36 24 16 40 24 20
1 16 0 24 0 40 0 0 0 -24 0 0 0 20 box5.dat
1 16 30 0 10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 10 0 10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 -10 0 10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 -30 0 10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 30 0 -10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 10 0 -10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 -10 0 -10 1 0 0 0 1 0 0 0 1 stud.dat
1 16 -30 0 -10 1 0 0 0 1 0 0 0 1 stud.dat
0

```

Kommando	Bedeutung	Erklärung
0	Kommentar	In der Zeile steht ein Kommentar. Dieses wird inzwischen auch für weitere Steuerbefehle benutzt.
1	Referenz	In dieser Zeile wird eine Referenz auf ein weiteres Legoteil eingefügt. Die Parameter umfassen die Transformationsmatrix und die Farbe für das neue Legoteil.
2	Linie	In dieser Zeile wird eine Linie definiert. Die Parameter sind die Farbe und die Koordinaten der Linie.
3	Dreieck	In dieser Zeile wird ein Dreieck definiert. Die Parameter sind die Farbe und die Koordinaten der Vertices des Dreiecks.
4	Viereck	In dieser Zeile wird ein Viereck definiert. Die Parameter sind die Farbe und die Koordinaten der Vertices des Vierecks.
5	optionale Linie	In dieser Zeile wird eine optionale Linie definiert. Dies ist eine normale Linie, die in LDraw aber nur angezeigt werden soll, wenn es sich um eine Linie handelt, die eine äußere Begrenzung des Legoteils repräsentiert.

Tabelle 1: Kommandos des DAT Formats

Früher wurden in den Kommentaren nur Informationen über den Autor der Datei ablegt. Inzwischen haben einige Entwickler jedoch erkannt, dass das DAT Format einige Schwächen hat und so werden Kommentare benutzt um Steuerbefehle für verschiedene Programme zu integrieren. So gibt es z.B. dem `STEP` Befehl, der anzeigt, dass alle vor diesem Kommentar auftretenden Referenzen und Geometrien zusammen gehören und angezeigt werden sollen. Damit ist es dann möglich ein Legomodell schrittweise aufzubauen, wie man es aus den Anleitungen der realen Modelle kennt. Auf die weiteren neuen Steuerbefehle wird an späterer Stelle eingegangen.

2.3 Probleme mit der Legobibliothek

Die Legobibliothek wurde seit 1995 immer weiter entwickelt. Viele Leute haben neue Legoteile erzeugt und diese wurden der Bibliothek hinzugefügt. Leider gab es keine Vorgabe, wie man die Modelle zu erstellen hatte, außer das LDraw sie anzeigen können mußte. Die Modelle in LDraw wurden aber immer nur ohne irgendeine Form der Beleuchtung dargestellt. Wenn man gerenderte beleuchtete Bilder eines Legomo-

dells haben wollte, mußte man einen Konverter bemühen, der ein Legomodell in ein POV-Ray Modell umwandelte und konnte dieses dann berechnen lassen. In den letzten Jahren wurden weitere Anzeigeprogramme erstellt, die 3D Hardware benutzen um die Modelle in Echtzeit anzuzeigen. Dabei sind dann aber einige Fehler in der Legobibliothek aufgefallen. Diese Fehler und wie sie in dieser Studienarbeit gelöst wurden, wird im folgenden beschrieben.

2.3.1 Orientierung der Flächen

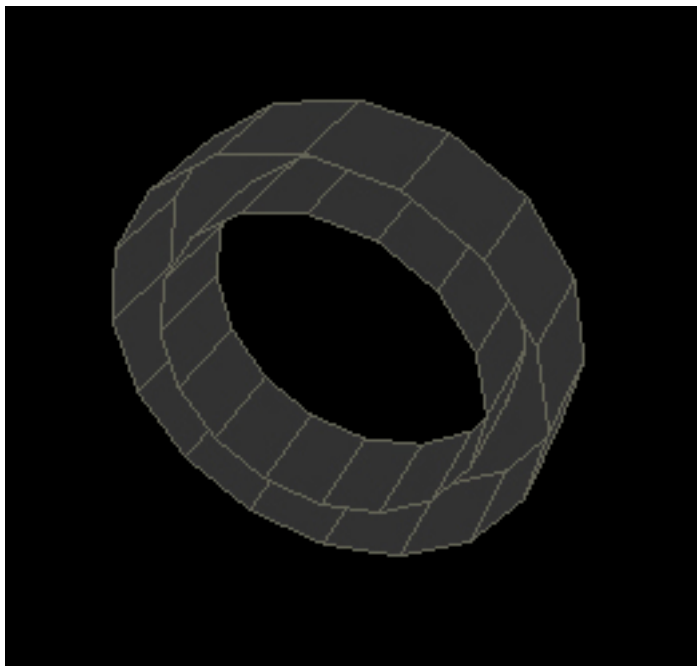


Abbildung 4: Innerer Studd - stud4.dat

In Abbildung 4 kann man einen `stud4` erkennen (Studds sind die kleinen Verbindungsstifte auf der Oberseite eines Legoblocks). Dieser Studd liegt auf der Innenseite eines Blocks und besteht aus zwei Zylindern und einer Scheibe auf der Oberseite. An diesem Modell kann man das Problem der Orientierung der Flächen erkennen. Die beiden Zylinder werden mit den gleichen Parametern importiert, nur verschieden skaliert. Innerhalb einer einzelnen Datei kann also nicht bestimmt werden welche Flächen nach innen oder nach außen zeigen, da es von der Datei abhängig ist, die dieses Modell dann importiert, wie die Flächen angeordnet sind.

Dadurch ist es nicht möglich die Vertices der Flächen so anzuordnen, dass man eine Vorder- und eine Rückseite hat. Um dennoch eine Echtzeitdarstellung der Modelle

mittels OpenGL zu erreichen, wurde **Two-Sided-Lighting** verwendet. In diesem Fall werden beide Seiten einer Fläche beleuchtet. Dieses geht natürlich auf Kosten der Performance, da nicht Flächen, die vom Betrachter wegzeigen nicht vorzeitig eliminiert werden können.

2.3.2 Schlechte Modellierung

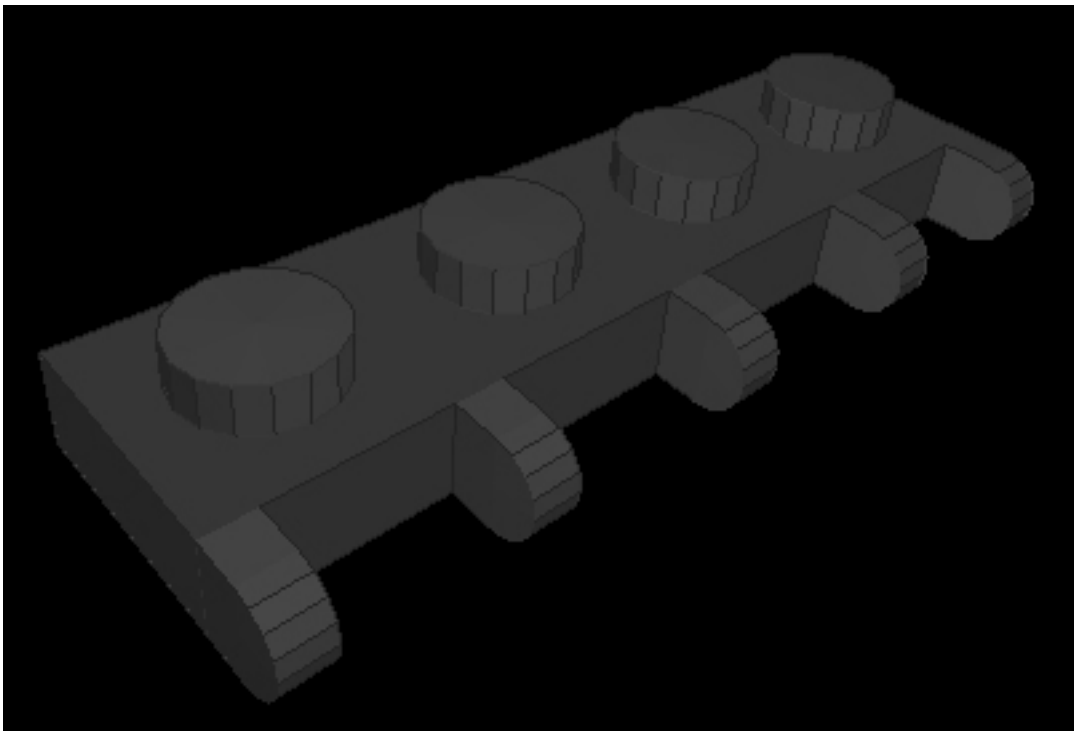


Abbildung 5: 4315.dat

Eine der früheren Modellierungsstrategien von LDraw Modellierern scheint „Hauptsache es wird angezeigt“ gewesen zu sein. Das in Abbildung 5 zu sehende Legoteil wurde zumindest auf diese Art und Weise erstellt. Die Transformationsmatrix für eine der Flächen, die die Halbzylinder an der Vorderseite seitlich abdecken, sieht folgendermaßen aus:

$$\begin{vmatrix} 0 & 0 & 0 & 40 \\ 4 & 0 & 0 & 4 \\ 0 & 0 & -4 & -14 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Wie man erkennen kann, wird die Fläche zuerst in X- und Z-Richtung gestreckt und dann einfach in die Y-Ebene „gedrückt“. dies hat zur Folge, dass keinerlei Normalen für diese Fläche berechnet werden können, da diese alle eine Länge von Null haben würden. Im unbeleuchteten LDraw war dies kein Problem. Die Lösung war die Matrix zu reparieren, indem in Zeilen, die nur aus Nullen bestehen Einsen eingefügt wurden, die eine korrekte Darstellung wieder erlauben. Die reparierte Matrix sieht dann wie folgt aus:

$$\begin{vmatrix} 0 & 1 & 0 & 40 \\ 4 & 0 & 0 & 4 \\ 0 & 0 & -4 & -14 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

2.3.3 Bowties

Nicht nur um die Orientierung der Flächen haben sich die Modellierer keine Sorgen gemacht, sondern auch die Anordnung der Vertices von Dreiecken und Vierecken. Bei Dreiecken macht es nichts aus, wie die Vertices angeordnet sind - zumindest so lange die Drehrichtung unerheblich ist, wie in unserem Fall, wo nicht bekannt ist, wo Innen und wo Außen ist. Bei Vierecken sieht dies jedoch anders aus. Es ist durchaus möglich, dass vier Vertices wie in Abbildung 6 angeordnet sind. In diesem Fall entsteht ein „Bowtie“ und die Darstellung ist nicht ein geschlossenes Viereck, sondern ein „Stundenglas“.

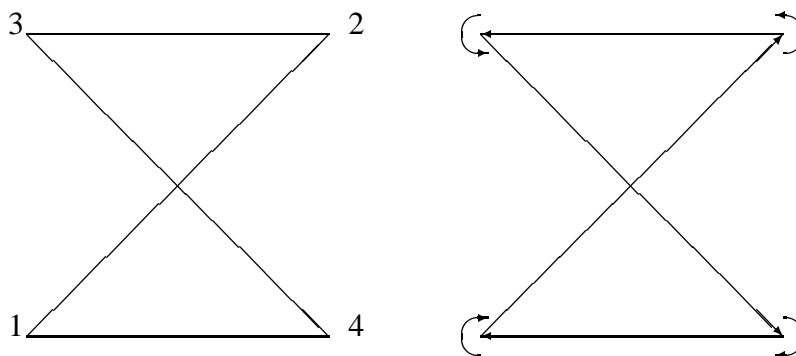


Abbildung 6: Bowties

Damit man diese Art von Vierecken vernünftig darstellen kann, muß man sie er-

kennen und reparieren. Erkennen kann man sie indem man die Differenzvektoren zwischen den Vertices bildet, wie auf der rechten Seite von Abbildung 6 zu sehen. Verfolgt man den Verlauf der Vektoren, gibt es an den Vertices Links-, bzw. Rechtsdrehungen. Ein Bowtie liegt vor, wenn es nicht nur ausschließlich Link- oder Rechtsdrehungen sind. Um welche Art Drehung es sich handelt kann man herausfinden, indem man die Normale in einem Vertex mit Hilfe der beiden an diesem Vertex inzidierenden Vektoren berechnet. Die Normalen von Links- und Rechtsdrehungen zeigen in verschiedene Richtungen. Nachdem man nun bestimmt hat, ob es sich um ein Bowtie handelt, muß es repariert werden. Dafür reicht es aus die Reihenfolge zweier Vertices auszutauschen, die die gleiche Drehrichtung haben.

3 Der LegoEditor

Aufbauend auf der Arbeit von Tomas Neumann wurde ein Editor entwickelt, dem eine Bibliothek von Objekten zur Handhabung von Legomodellen als Basis dient. Diese Bibliothek kann auch in anderen Projekten verwendet werden um Legomodelle anzuzeigen.

3.1 Optimierungen

3.1.1 Two Sided Lighting

Der GLegoViewer von Tomas Neumann war in der Lage Legomodelle anzuzeigen. Allerdings zeigte sich, dass größere Modelle nur sehr langsam angezeigt werden konnten. Das Problem war die ungeheure Anzahl an Polygonen die ein schon recht kleines Modell besitzt. So besitzt das Auto `car.dat` ca. 13000 Polygone, der in Abbildung 7 zu sehende Snowspeeder ca. 65000 Polygone. Da nicht bekannt ist, welche Seite einer Fläche die Innen- oder Außenseite des Modells ist, hat GLegoviewer alle Flächen verdoppelt und aufeinandergelegt, so dass es immer eine entsprechende Fläche gezeichnet werden konnte. Dadurch hat sich der Auswand natürlich verdoppelt.

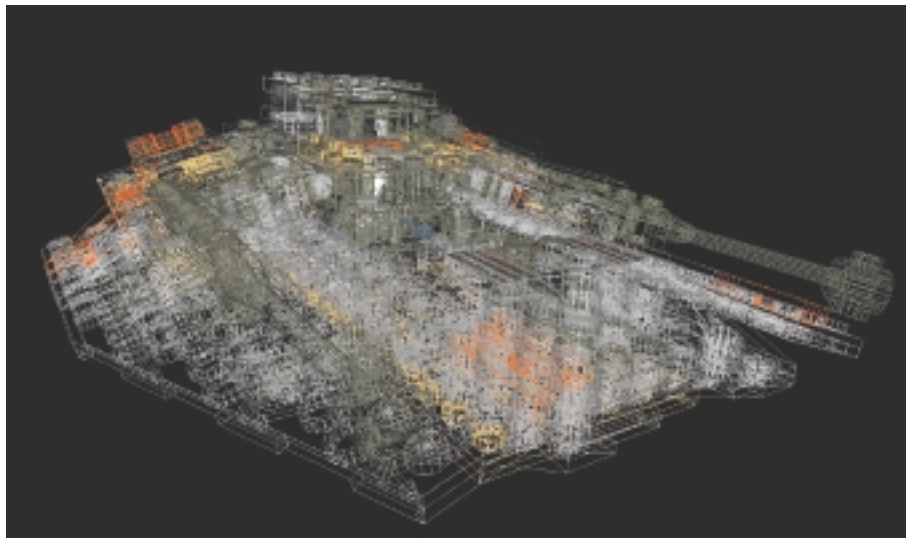


Abbildung 7: Snowspeeder in Wireframe-Darstellung

Durch `Two Sided Lighting` war es möglich Vorder- und Rückseiten der Polygone zu beleuchten. Somit machte es keinen Unterschied mehr, ob eine Fläche eine Innen- oder Außenseite war. Natürlich ist dies noch nicht optimal, da immer noch alle

Flächen gezeichnet werden und nicht sichtbare Flächen, die vom Betrachter wegweisen, nicht weggelassen werden können.

3.1.2 Vertex Arrays

GLegoViewer verwendete OpenGL Displaylisten um die einzelnen Objekte zu zeichnen. Diese wurden durch Vertexarrays ersetzt. Für jedes Objekt wurde ein Vertexarray angelegt. Zusätzlich zum Vertexarray gibt es ein Array der Normalen in jedem Vertex. Dadurch ist es möglich einzelne Flächen durch Smoothshading zu schattieren.

3.1.3 Entfernen der Rekursion

Da ein Legomodell hierarchisch aufgebaut ist, liegt es nahe das Modell rekursiv zu rendern. Jeder Knoten im Baum ist in der Lage die eigene Geometrie anzuzeigen und seine Kinder aufzufordern das gleiche zu tun. Im Verlauf der Entwicklung des LegoEditors hat sich aber gezeigt, dass dieses rekursive Rendern sehr viel Zeit verschlingt. Also wurde eine Möglichkeit implementiert Vertexarrays zu kombinieren. So war es einem Knoten im Baum möglich in sein Vertexarray alle Geometrieinformationen seiner Kinder aufzunehmen und die auf einen Schlag zu zeichnen. Dadurch lies sich ein Geschwindigkeitszuwachs vom Faktor 10 erzielen.

3.2 Die Lego API

Die Legobibliothek besteht aus drei grundlegenden Objekten:

- `LegoObject`
- `LegoModelData`
- `LegoScene`

Diese drei Objekte können in anderen Projekten verwendet werden. Allerdings benötigen sie zur Darstellung der Objekte OpenGL.

3.2.1 LegoObject

Ein `LegoObject` kapselt alle Informationen über ein einzelnes Legoteil, welches in einer Datei gespeichert ist. Alle Geometrieinformationen und Referenzen auf andere

Objekte sind in ihm gespeichert. Es ist in der Lage per OpenGL die vorhandene Geometrie zu rendern. Das Objekt beinhaltet auch die nötigen Vertex- und Normalarrays.

3.2.2 LegoModelData

LegoModelData bildet eine Ablage für LegoObjects. Diese Klasse ist in der Lage Legomodelldateien zu laden. Diese werden dann abgelegt und stehen fortan zur Verfügung. Sollten mehrere Dateien die gleichen Untermodelle benutzen, wird jedes Modell nur einmal geladen. Jedes Objekt bekommt eine eindeutige Nummer zugewiesen über die es andere Modelle referenzieren können. Ein LegoModelData-Objekt übernimmt beim Laden die Erzeugung der LegoObject-Objekte. Die Dateien werden zeilenweise eingelesen und analysiert. Geometrieinformationen werden in den neu erzeugten LegoObjects ablegt und Referenzen auf weitere Dateien werden vermerkt, falls diese noch nicht gelesen wurden, damit dies später automatisch geschehen kann. Wenn ein LegoObject komplett gelesen wurde, werden Vertex- und Normalenarray erzeugt und gespeichert. Die LegoModelData-Klasse kann auf einzelne LegoObject-Objekte optimieren, indem ein neues LegoObject erzeugt wird und die Vertexarrays aller Untermodelle in diesem Objekt kombiniert werden. Eine LegoModelData-Instanz reicht aus, um Legomodelle auf dem Bildschirm sichtbar zu machen, da jedes LegoObject in der Lage ist sich selber darzustellen. Es reicht also sich aus dem Vorrat der abgelegten Objekte eines herauszugreifen und dessen `render()` Methode aufzurufen.

3.2.3 LegoScene

Die LegoScene-Klasse erlaubt es mehrere Legomodelle auf einmal anzuzeigen, neue Modelle der Szene hinzuzufügen, alte zu löschen oder Modell bzw. Untermodelle durch den Raum zu schieben. Eine Szene besteht dabei aus einem Baum von `LegoScene::LegoSceneNode`-Objekten, die jeweils ein einzelnes Legomodell in der Szene repräsentieren. Jedes Blatt und jeder Knoten im Baum verweist auf ein `LegoObject` in der Modellablage der Szene. Die `LegoObjects` werden also nur einmal angelegt und mehrfach verwendet.

3.3 Der LegoEditor

LegoEditor ist eine Beispielanwendung für die Funktionen der entwickelten Lego API. Er bietet die Möglichkeit Legomodelle zu laden und zu betrachten. Da er ein `LegoScene`-Objekt zur Grundlage hat (dessen Inhalt auf der linken Seite in einem Baum dargestellt wird), kann man natürlich mehrere Modelle laden und diese im Raum verschieben. Als kleinste veränderliche Einheit wurde dabei ein Legoteil gewählt, da ein Editor für Szenen entwickelt werden sollte und kein Modeller für einzelne Legoteile. Man kann also aus einem Modell ein Legoteil auswählen und es an einen anderen Ort schieben oder dies auch mit einem kompletten Modell machen. Verschiedene Betrachtungsmodi erlauben dem Benutzer ein Gefühl für die hohe Komplexität eines Legomodells zu entwickeln. Außerdem ist es möglich ein Modell schrittweise aufzubauen, wenn die entsprechenden Informationen in den Modelldateien vorhanden sind. Entwickelt wurde der Editor unter FLTK 1.1.2. Der Baum auf der rechten Seite ist ein Extrawidget, welches unter [3] zu finden ist. Der Editor nutzt einige Möglichkeiten von FLTK zur Nachrichten- und Callbackweiterleitung aus und kann einfach als Vorlage für neue Projekte genutzt werden.

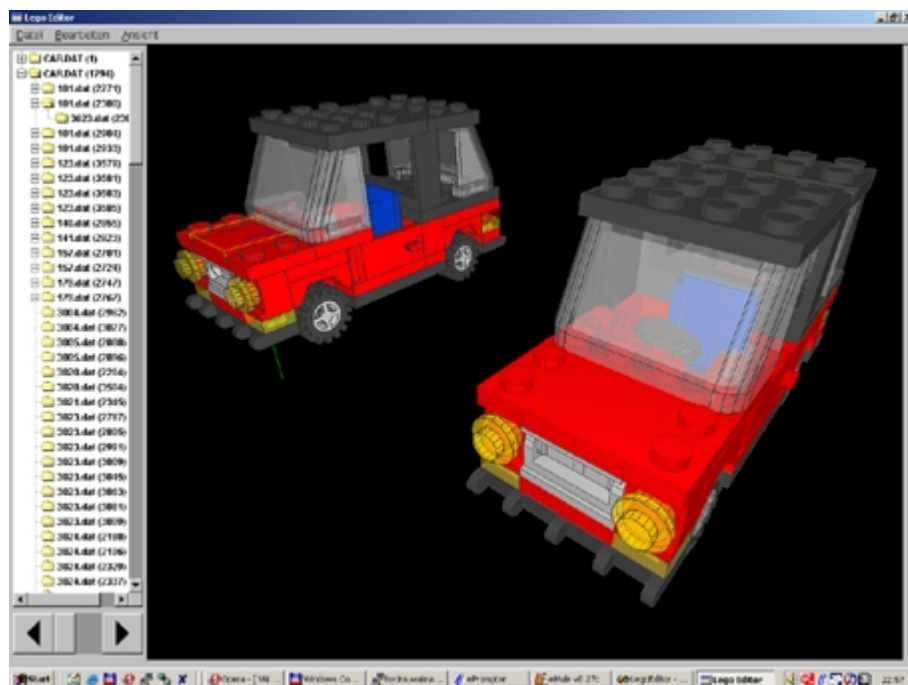


Abbildung 8: LegoEditor

3.4 Weitere Optimierungen

Mit der Lego API und dem LegoEditor wurde die Ausgabe der Legomodelle zwar erheblich beschleunigt, es gibt aber immer noch Möglichkeiten die Darstellung weiter zu optimieren. So werden immer noch alle Flächen, des Modells gezeichnet. Dieses Problem steigert sich natürlich mit der Anzahl verwendeter Legoteile. Ein erster Schritt wäre es also zu erkennen, welche Fläche einen Innenseite und welche eine Außenseite ist. Inzwischen gibt es Bestrebungen innerhalb der LDraw-Community solche Informationen in den Modelldateien abzuspeichern. Allerdings gibt es inzwischen sehr viele Dateien und es gibt keinen automatischen Mechanismus um dies zu bestimmen.

Die nächste Möglichkeit zur Beschleunigung bestünde darin innere Geometrien, die garantiert niemals angezeigt werden zu entfernen. Wenn man zwei Legosteine aufeinander steckt, verschwinden die Stifte, die die beiden Teile zusammenhalten. Diese müssen nie wieder angezeigt werden. Ein Studd (`stud4.dat`) besteht immerhin aus 16 Dreiecken und 16 Vierecken. Wenn man dies auf ein gesamtes großes Modell überträgt, ist nicht schwer zu verstehen, was eine solche Beschleunigung bewirken würde.

Weitere Optimierungen könnten im Bereich der Darstellungsqualität vorgenommen werden. Die Modelle werden inzwischen zwar beleuchtet, werfen aber noch keinen Schatten. Und in Zeiten moderner 3D-Hardware sollte es auch möglich sein, dass sich Legomodelle selbst schattieren.

Da Legoteile immer eine gewisse Größe haben, meistens rechteckig sind und die Stifte auf bestimmten Positionen liegen, sollte es auch möglich sein einen Editor zu konstruieren, der die Legoteile an einem Raster ausrichtet und blockierte Positionen sperrt, so dass man Legosteine nur in physikalisch Mögliche Art und Weise anordnen kann.

Eine weitere Anwendung wäre ein Konverter, der ein beliebiges Modell nimmt und es in ein Legomodell umwandelt, indem er z.B. nur kleine Legosteine verwendet und mit ihnen das vorgegebene Volumen füllt.

Literatur

- [1] LDraw.org
<http://www.ldraw.org>
- [2] LDraw.org - Spezifikation des DAT Formats
<http://www.ldraw.org>
- [3] FLU - FLTK Utility Widgets
<http://www.osc.edu/~jbryan/FLU/>