Herausforderungen an moderne Szenengraphsysteme am Beispiel OpenSG

Dirk Reiners

Die Entwicklung von Computerhardware schreitet unverändert rasant voran, Moore's Law hat nichts von seiner Aussagekraft verloren. Aber es gibt Bereiche innerhalb der Computerhardware, die sich noch wesentlich schneller entwickeln. Einer dieser Bereiche ist die Hardware für dreidimensionale Computergraphik.

Hier erfolgte die Leistungs- und Komplexitätsentwicklung in den letzen Jahren doppelt so schnell wie Moore's Law vorhersagt, was dazu führt, dass Anwendungen, die noch vor kurzem Millionen EUR teuren Graphiksupercomputern vorbehalten waren, auf handelsüblichen PCs, den sprichwörtlichen Aldi-Rechnern, realisiert werden können. Die Entwicklung im Consumer-Segment war so rasant, dass sie die Entwicklung im High-End-Bereich überflügeln konnte (Abb. 1). Nach 2001 hat sich der Schwerpunkt von der Transformationsleistung auf Qualität verlagert, aber die Entwicklung schreitet unverändert schnell voran. Dies hat bei den Workstation-Herstellern dazu geführt, dass sie die Entwicklung eigener Graphikhardware weitgehend eingestellt haben und statt dessen die Chips der PC-Hersteller wie nVidia und ATI verwendet werden.

Eine Konsequenz dieser enormen Preissenkung im Hochleistungsgraphikbereich ist, dass mehr und mehr verschiedene Anwender und Anwendungsgruppen Zugriff auf diese Technologie haben. Diese Anwender kommen sowohl aus dem universitären als auch aus dem wirtschaftlichen Bereich.

An den Universitäten wird jetzt wesentlich mehr Forschung im Bereich interaktive und Echtzeitgraphik betrieben, da die früher benötigten Spezialrechner mit ihren enorm hohen Einstiegskosten weggefallen sind. Dies führt auch dazu, dass vermehrt neue Erkenntnisse gewonnen und Methoden in der Forschung entwickelt werden können, die völlig neue Möglichkeiten eröffnen.

Im wirtschaftlichen Bereich denken vermehrt Unternehmen darüber nach, wie sie die aktuellen Entwicklungen für sich sinnvoll einsetzen können. Diese betrifft die einfache Benutzung der jetzt preisgünstiger verfügbaren Hardware, aber auch Umsetzung und Einsatz aktueller Forschungsergebnisse.

Ein sich daraus ergebendes Problem ist die Kombination der unterschiedlichen Anforderungen dieser beiden Gruppen. Forschungsgruppen entwickeln ihre Verfahren, so dass sie möglichst flexibel verwendet und leicht verändert werden können, um neue Erkenntnisse zu gewinnen. Anwender wollen möglichst wenig mit den Details belastet werden, und die möglichen Verfahren einfach nur nutzen.

Die Frage dabei ist, wie man diese unterschiedlichen Anforderungen integrieren kann, und im Bereich Computergraphik gibt es dafür ein wichtiges und weit verbreitetes Konzept: Szenengraphen.

Szenengraphen

Motivation

Graphikhardware wird typischerweise über ein Low-level-API wie OpenGL [6] oder Direct3D [5] angesprochen. Das Ziel dieser Interfaces liegt darin,

DOI 10.1007/s00287-004-0440-7

D. Reiners
(dirk@opensg.org) OpenSG Forum im ZGDV,
Fraunhoferstr. 5,
64283 Darmstadt

D. Reiners
Virtual Reality Applications Center,
Iowa State University,
1620F Howe Hall, Ames, IA 50011–2274, USA
E-Mail: dreiners@iastate.edu

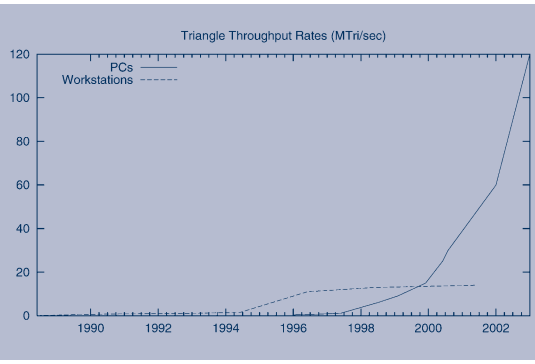

Abb. 1 Entwicklung der Polygontransformationsleistung im PC- und Workstationbereich

möglichst dicht über der Hardware zu liegen und die konkrete Implementierung zu abstrahieren, um Programme auf den Systemen verschiedener Hersteller verwenden zu können. Gleichzeitig sollen sie nicht so komplex werden, dass sie unnötigen Aufwand verursachen oder die Entwickler in ihrer Freiheit einschränken. Daher sind diese Systeme sehr einfach und auf niedrigem Abstraktionsniveau.

Sie betrachten die darzustellenden Daten als einen Strom von nur minimal zusammenhängenden Dreiecken, sehen aber nicht die logischen Zusammenhänge von Objekten. Daher muss der Entwickler für die Bilderzeugung in diesen Systemen auch für jedes Bild sämtliche Daten neu spezifizieren, was bei komplexen Algorithmen sehr mühsam werden kann. Insbesondere wenn die schnelle Darstellung komplexer Modelle erwünscht ist, muss der Entwickler sehr viel Zeit und Erfahrung mitbringen.

Graphikhardware entwickelt sich nicht nur bedeutend schneller als Standardprozessoren, sie ist inzwischen auch wesentlich komplexer geworden. Während aktuelle Prozessoren wie Intel's Pentium 4 etwa 55 Millionen Transistoren beinhalten, liegen aktuelle Graphikchips wie nVidias GeForce FX 6800 oder ATIs Radeon x800 bei dem 3- bis 5fachen Wert. Dazu kommt noch, dass bei Standardprozessoren ein Großteil der Transistoren in den einfach und repetitiv aufgebauten Caches liegt. Graphikprozessoren haben vergleichsweise wenig Speicher auf dem Chip, und stattdessen wesentlich mehr als Pipeline-organisierte komplexe Recheneinheiten. Diese Pipeline kann bei falscher Benutzung schnell gestört werden, was zu Flushes führt, die einen großen Teil der erreichbaren Leistung vernichten kann. Scheinbar harmlose Kommandos wie das Ändern der Linienbreite am falschen Ort können Performanzverluste bis zu einem Faktor 100 oder mehr verursachen.

Szenengraphsysteme

3D-Graphikhardware ist inzwischen überall und billig verfügbar. Dies weckt Interesse zum einen bei neuen Entwicklern von Verfahren, aber auch bei neuen Anwendern. Szenengraphen sind ein guter Weg, die beiden Gruppen zusammen zu bringen. Sie erlauben die Beschreibung der graphischen Daten auf für Anwender verständliche Weise, ohne dass er sich um Details und Beschränkungen der Hardware kümmern muss. Analog zu den Programmiersprachen gilt es, eine objektoriente Ebene à la C++ oder Java auf die vorhandenen, Assembler-ähnlichen Schnittstellen aufzusetzen.

Sie sind keine neue Idee, es gibt diverse kommerzielle und Open-Source-Systeme. Aktuelle Entwicklungen bringen aber neue Herausforderungen mit sich. Dieser Beitrag beschreibt die Herausforderungen in den Bereichen Laufzeiterweiterbarkeit, Unterstützung aktueller Graphikhardwarefeatures und Graphikalgorithmen, Unterstützung von Multithreading und Clustering und wie sie innerhalb von OpenSG gelöst wurden.

Für Entwickler, für die die Graphik nur ein Mittel zum Zweck und nicht Selbstzweck ist, ist das eine zu niedrige Abstraktionsebene mit zu vielen Problemen. Diese Entwickler sehen eher die auf einem höheren Niveau liegende Objektstruktur ihrer Anwendung, in der ganze Objekte oder Objektgruppen manipuliert werden. Diese sollen möglichst nur einmal spezifiziert und dann wiederverwendet werden. Der Aufwand zur Erzeugung neuer Bilder mit allen verfügbaren Features sollte möglichst minimal sein, im besten Fall über ein einziges Kommando erfolgen können.

Übertragen auf Programmiersprachen arbeiten Direct3D und OpenGL auf Assemblerniveau. Genauso wie heutzutage nur noch ein kleiner Teil der Entwicklung in Assembler erfolgt, muss für die Entwicklung interaktiver Graphiksysteme eine höhere Ebene zur Verfügung gestellt werden, die den höheren, objektorientierten Sprachen wie C++ entspricht. Eine Möglichkeit zur Strukturierung einer solchen Softwareebene ist das Prinzip des Szenengraphen.

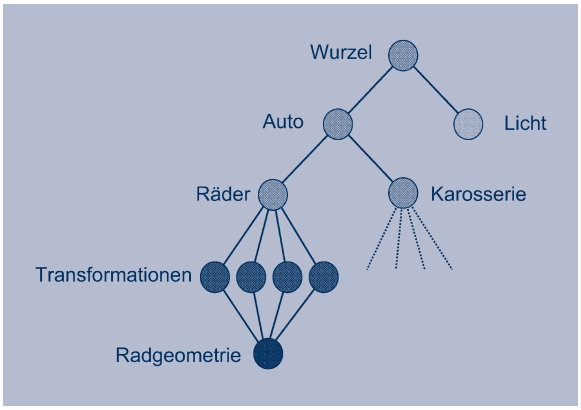

Abb. 2 Szenengraphstruktur. Alle Kanten sind gerichtet und Zyklen sind nicht erlaubt. Verschiedene Farben repräsentieren verschiedene Arten von Knoten

Graphic Utilities

3D graphics hardware is becoming ubiquitous and cheap. This attracts the attention of new developers as well as new users. Scenegraphs are a good way of getting the two groups together. They describe the graphical data on a level that is more understandable for users without having to worry about details and limitations of the underlying hardware. Similar to programming languages the goal is to add an object-oriented, C++ or Java-like layer on top of the exisiting interfaces, which are much closer to assembly language.

They are not a very new idea, there are a number of commercial and Open Source systems available. However, new developments come with new challenges. This article describes the challenges in the areas of runtime extensibility, support of new graphics hardware features and algorithms, as well as support for multi-threading and clustering, and how they were solved in the OpenSG system.

Struktur

Graphen sind in der Informatik weit verbreitet und werden für viele verschiedene Anwendungsgebiete verwendet. Szenengraphen sind eine spezielle Gruppe von Graphen mit folgenden Eigenschaften: Sie sind

- gerichtet,
- azyklisch,
- heterogen.

Abbildung 2 zeigt ein Beispiel für einen typischen, kleinen Szenengraphen. Dabei kann der Aufbau von oben nach unten oder von unten nach oben erfolgen. Im Beispiel bilden die Räder und die Karosseriebestandteile (nicht dargestellt) die Blätter. Typisch für Szenengraphen ist die mehrfache Verwendung von Teilbäumen (hier die Radgeometrien), um Speicherplatz zu sparen. Um die verschiedenen Instanzen der Radgeometrien an die richtigen Stellen am Fahrzeug zu bewegen, dienen Transformationsknoten, deren Transformationen auf alle ihre Nachkommen im Baum angewendet werden. Durch die Verwendung mehrerer Transformationsknoten lassen sich so sehr leicht hierarchische Transformationen (wie Planeten- und Mondsysteme, oder mehrgelenkige Roboterarme) definieren und manipulieren. Innere Gruppenknoten erlauben eine logische Strukturierung der Objekte im Baum, so lässt sich z. B. durch Manipulationen am Auto-Objekt das gesamte Fahrzeug bearbeiten. Zur Vervollständigung der Gesamtszene werden noch weitere Knoten wie Lichtquellen etc. benötigt. Das Gesamtsystem wird schließlich durch den Wurzelknoten spezifiziert.

In der Heterogenität, also dem Vorhandensein von verschiedenen Typen von Knoten, liegt ein zentraler Bestandteil der Flexibilität und Ausdrucksfähigkeit eines Szenengraphen. Abbildung 3 zeigt einen Überblick über die Knotentypen in OpenSG. Generell lassen sich die Typen in innere Knoten und Blätter zerlegen, bei OpenSG sind erstere von Group abgeleitet (linke Seite von Abb. 3), letztere von Drawable (rechte Seite von Abb. 3). Innere Knoten definieren die Struktur des Baums und legen fest, welche Teilbäume als logische Objekte angesehen werden. Manche Knotentypen definieren auch Attribute, die an alle Kinder vererbt werden, wie z. B. oben genannte Transformationen. Die eigentlichen geometrischen Informationen der darzustellenden Objekte sind meist in den Blättern gespeichert. Alle Szenengraphsysteme unterstützen hier polygonale, also aus einzelnen, ebenen Flächenstücken zusammengesetzte Geometrien, da diese von der Graphikhardware direkt unterstützt werden. Moderne Systeme bieten hier mehr Vielfalt, um die Entwicklung komplexer Applikationen zu vereinfachen.

Operationen auf den Graphen sind als Traversierungen des Graphen realisiert. Verschiedene Traversierungsvarianten lassen sich dabei bei fast allen Szenengraphsystemen wiederfinden. An erster

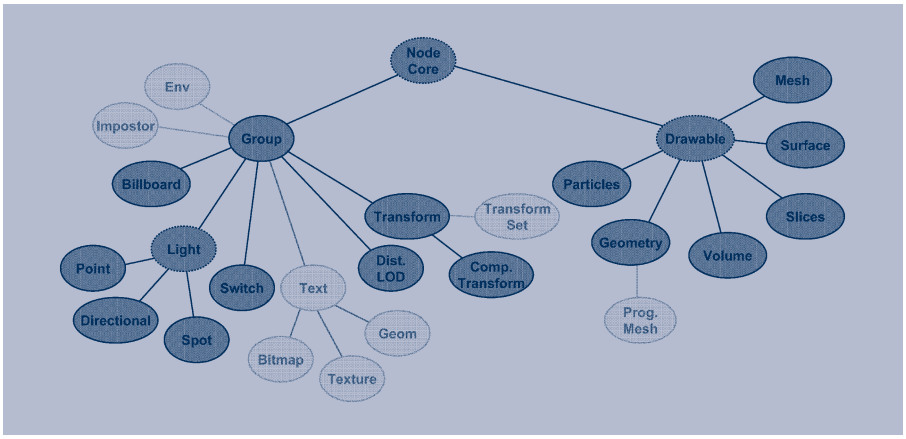

Abb. 3 OpenSG-Knotentypen. Helle Typen sind in Entwicklung

Stelle steht hier natürlich die Darstellungstraversierung, die die im Graph gespeicherten Daten in ein neues Bild umsetzt. Viele Systeme unterstützen auch den Test der Geometrie gegen einen Strahl, für Interaktion oder Kollisionserkennung. Ebenfalls zu finden sind Traversierungen um den Graph zu drucken oder in eine Datei auszugeben, oder um ihn in eine effizienter darzustellende Version zu transformieren. Auch die Anwendungslogik wird oft in Traversierungen ausgedrückt.

Existierende Systeme

Die Grundidee des Szenengraphen ist nicht neu, erste Systeme wurden bereits Anfang der 90er-Jahre entwickelt. Open Inventor [15] und OpenGL Performer [13] von SGI sind die bekanntesten Vertreter, andere sind Java3D [16], Cosmo3D und Optimizer. Die 90er waren eine wechselhafte Zeit für Szenengraphen (s. Infobox: Die Wilden 90er).

Die Wilden 90er

Szenengraphen kamen Anfang der 90er-Jahre auf. Silicon Graphics (heute SGI) sah sich gezwungen, zu ihren Graphikrechnern Softwaresysteme zur Verfügung zu stellen, die auch unerfahrenen Anwendern erlaubten, die komplexen Systeme optimal zu nutzen. Open Inventor und OpenGL Performer waren die ersten Vertreter.

Beide sollten von dem 1995 vorgestellten Cosmo3D abgelöst werden, zu dem 1996 Optimizer als Ergänzung hinzukam. Optimizer war das erste System mit direkter Freiformflächenunterstützung, was es für viele Anwendungen in der Konstruktion prädestinierte.

Cosmo3D war in Zusammenarbeit mit Sun begonnen worden, was aber gegen Ende 1996 in eine Sackgasse führte. Als Ausweg wurde versucht, unter der Ägide des OpenGL Architecture Review Boards ein neues Szenengraphsystem, OpenGL++, zu etablieren. Dieses überlebte aber nicht einmal das Jahr 1997, als es vom Fahrenheit-Projekt abgelöst wurde.

Fahrenheit war der Versuch von sgi, mit Microsoft zusammen ein High-Level-Graphikinterface zu definieren, das Anforderungen einer Anwendungspalette von Spielen bis zu CAD-Systemen gerecht werden sollte. Es wurden viele interessante Versprechungen gemacht, allerdings wurde die Zusammenarbeit noch vor der ersten Version im Sommer 1999 abgebrochen. Microsoft hat dann noch eine Version 1.0 herausgebracht, aber unter der Prämisse, dass keine Weiterentwicklung und kein Support geleistet werden kann.

Nachdem die Entwicklergemeinde über mehr als 4 Jahre hingehalten worden war, wandte sich das Interesse stark von kommerziellen Systemen ab, hin zu Open-Source-Systemen wie OpenSG.

Zu Beginn des 21. Jahrhunderts haben Open-Source-Szenengraphsysteme massiv an Bedeutung gewonnen. Die bekanntesten Vertreter sind Open Scenegraph [9] und OpenSG [10], weitere Systeme sind Coin ([2], ein Open Inventor Klon), der originale Open Inventor ([7], der von sgi freigegeben wurde), SGL [14], OpenRM [11] und viele andere.

Die meisten heute gebräuchlichen Szenengraphsysteme sind C++-Bibliotheken. Von einigen existieren noch C-Versionen, und manche haben alternative Anbindungen an Scriptsprachen wie Lua oder Python, aber generell ist C++ die meistbenutzte Programmiersprache für interaktive Graphikanwendungen.

Herausforderungen

Szenengraphsysteme werden in vielen verschiedenen Anwendungsgebieten eingesetzt. Überall wo interaktive oder echtzeitbasierte 3D-Graphik gebraucht werden, kann es Sinn machen, auf einem Szenengraph aufzusetzen. Viele derartige Anwendungen werden unter dem Titel virtuelle Realität (VR) zusammengefasst. Die Entwicklung dieser VR-Anwendungen bedingt heutzutage Herausforderungen für den Szenengraph in verschiedenen Gebieten:

- Laufzeiterweiterbarkeit,
- Unterstützung neuer Graphikhardware,
- Unterstützung aktueller Graphikalgorithmen,
- Unterstützung von Multithreading,
- Unterstützung von Clustering.

Laufzeiterweiterbarkeit

Wie schon erwähnt beziehen Szenengraphsysteme einen Großteil ihrer Flexibilität und Stärke aus der Vielfalt der Knotentypen. Es kann Sinn machen, spezielle Knoten oder spezielle Traversierungen je nach Applikation hinzuzufügen. Diese können entweder ganz neue Knoten und Fähigkeiten hinzufügen, oder vorhandene ersetzen. Beides sollte möglich sein, ohne den Quellcode des Szenengraphen zu verändern, da sonst diese applikationsspezifischen Veränderungen bei jeder neuen Version des Szenengraphen nachgezogen werden müssen, was bei der generell schnellen Versionsfolge von Open-Source-Systemen einen hohen Aufwand bedeutet.

Neue Knotentypen

Die primäre Aufgabe beim Hinzufügen neuer Knotentypen ist die Erweiterung der Traversierungen. Diese müssen in der Lage sein, die neuen Knoten im Zusammenhang mit den schon vorhandenen zu bearbeiten.

Objektorientierte Sprachen wie C++ stellen hierfür geeignete Techniken in Form von virtuellen Methoden zur Verfügung. Szenengraphen stellen ein Paradebeispiel für objektorientierte Programmierung dar. Die Basisklasse der Knoten bekommt dabei eine virtuelle Methode für jede Traversierung, die von den verschiedenen konkreten Klassen überladen wird. Dieses Vorgehen ist so natürlich, dass es in vielen Einführungskursen zur Computergraphik als Aufgabe verwendet wird, und es erfüllt die oben genannte Anforderung, dass Erweiterungen ohne Änderungen am Quellcode der Szenengraphbibliothek möglich sind.

Neue Traversierungen

Da der größte Teil der Operationen auf einem Szenengraph mittels Traversierungen durchgeführt wird, ist das Hinzufügen neuer Traversierungen der natürliche Weg für Anwendungen, ihre Operationen zu realisieren.

Unter Verwendung der oben skizzierten Methode bedeutet dies das Hinzufügen einer neuen virtuellen Methode zur Knotenbasisklasse, die dann in den verschiedenen konkreten Knotentypen überladen wird. Der Nachteil dieses Ansatzes für Traversierungen ist, dass er Änderungen am Quellcode der Bibliothek erfordert.

Ein alternativer Ansatz, der dieses Problem nicht hat, findet sich unter den bekannten Design Patterns [3]: das Visitor Pattern. Dabei wird die virtuelle Methode aus den Knotenklassen in separate Visitorklassen verlagert, die dann von den Knoten aus aufgerufen werden.

Dieser Ansatz löst das Problem für neue Traversierungen, aber dafür müssen nun zum Einfügen neuer Knotentypen die vorhandenen Traversierungsobjekte geändert werden. Eine Lösung, die beide Erweiterungen erlaubt, muss daher anders aussehen.

Allgemeine Lösung

Das fundamentale Problem bei den beiden oben genannten Ansätzen ist, die Abbildung von konkreten Klassen auf auszuführenden Methoden durch den Compiler entweder für die Knotenklassen oder die Visitorklassen durchzuführen. Beide sind integrale Bestandteile der Bibliothek, weswegen Änderungen und Erweiterungen in den Quellcode der Bibliothek hineinreichen.

Um applikationsseitige Erweiterungen auf beiden Seiten zu erlauben, muss diese Abbildung aus den Kernklassen herausgelöst werden. OpenSG löst das Problem durch eine explizite, dynamisch erweiterbare Abbildung der traversierten Knotenklasse auf die auszuführende Methode. Jede Klasse in OpenSG hat Informationen über ihren Typ. Die Informationen werden benutzt, um ein Funktorobjekt zu selektieren. Diese Funktorobjekte sind so realisiert, dass sie sowohl eine Methode des traversierten Objekts als auch Methoden unabhängiger

Objekte oder einfache Funktionen referenzieren können. Dadurch können sie das ursprüngliche Verhalten, bei dem eine virtuelle Methode des Knotens aufgerufen wird, emulieren, ohne darauf festgelegt zu sein.

Neue Knotentypen können einfach zur Laufzeit bei den vorhandenen Traversierungen angemeldet werden. Neue Traversierungen benötigen eine Liste von Funktoren für die eingebauten Knotentypen, die – wie oben erwähnt – jedoch nicht Methoden der Knotentypen referenzieren müssen (aber können).

Dieses Vorgehen erlaubt auch, das Verhalten eingebauter Traversierungen für eingebaute Knotentypen zur Laufzeit zu verändern (durch Austausch des eingebauten Funktors durch einen benutzerdefinierten), und erlaubt damit die größtmögliche Flexibilität.

Ersetzung eingebauter Typen

Das Ersetzen des Verhaltens der eingebauten Typen bei Veränderungen ist ein erster Schritt in Richtung eines flexiblen und offenen Systems, aber alleine nicht ausreichend. Wenn Veränderungen nicht nur neue Methoden, sondern auch neue Daten erfordern, muss mehr getan werden. OpenSG unterstützt dies durch zwei Mechanismen.

Erzeugung: Die Grundlage der Ersetzung eingebauter Typen ist die Entkoppelung des logischen vom konkreten Typ. Die Anwendung fordert die Bibliothek auf, einen Knoten eines bestimmten Typ zu erzeugen, z. B. eine Geometrie, der konkrete Typ bleibt aber der Bibliothek überlassen, und kann durchaus ein anderer, von der Geometrie abgeleiteter sein. Hier bieten die bekannten Design Patterns [3] eine gute Lösung in der Form des Prototype Patterns an. Die Erzeugung neuer Instanzen geschieht dabei nicht durch explizite Erzeugung, sondern durch Replikation einer Prototypinstanz. Diese kann von einem beliebigen, kompatiblen Typ sein und ist nicht auf bereits in der Bibliothek bekannte Typen limitiert. Diese Veränderungen wirken sich dann auf alle danach erzeugten Objekte aus. Dies beinhaltet nicht nur die von der Anwendung neu erzeugten Objekte, sondern schließt die von der Bibliothek erzeugten Objekte (z. B. durch Laden einer Datei) ein.

Ein weiterer Vorteil des Prototype Patterns ist die Verwendung einer konkreten Instanz als Prototyp. Dies erlaubt die Veränderung des initialen Zustands neu erzeugter Objekte einfach durch Manipulation des Prototypzustands, der dann in alle danach erzeugten Objekte dieses Typs kopiert wird.

Diese Erzeugungsmodalitäten geben der Anwendung weitreichende Möglichkeiten der Manipulation interner Typen der Bibliothek. Um der Bibliothek eine Möglichkeit zu geben, mit dieser Unsicherheit umzugehen, benötigt sie Informationen über die Eigenschaften der Klassen, die über die vom C++-Standard definierten hinausgehen.

Reflektivität bezeichnet in diesem Kontext die Fähigkeit von Datenstrukturen, über sich selbst Auskunft geben zu können. Datenstrukturen in OpenSG können in abstrahierter Weise mitteilen, welche Datenfelder sie besitzen, welchen Typ und welche Kardinalität die abgelegten Daten haben und wie auf diese Daten in generischer Form (ASCII und binär) zugegriffen werden kann.

Diese Fähigkeiten werden benutzt um z. B. generische User-Interface-Komponenten zu entwickeln (Abb. 4). Diese Komponenten können beliebige OpenSG-Klassen darstellen und manipulieren. Dies schließt auch Klassen mit ein, die in der Anwendung nach Compilation der Bibliothek hinzugefügt wurden. Die gleichen Mechanismen werden auch verwendet, um Multithreading und Clustering in generischer Weise zu unterstützen. Anwendungsspezifische Klassen werden also automatisch in Multithreading und Clustering mit eingeschlossen, ohne dass der Anwendungsentwickler dafür einen speziellen Programmcode schreiben müsste.

Unterstützung neuer Graphikhardware

Graphikhardware wird nicht nur in kurzen Abständen schneller, sondern gewinnt mit jeder Generation auch an neuer Funktionalität. Da diese abhängig vom Hersteller und gelegentlich auch vom Modell sind und teilweise relativ kleine Nischengebiete abdecken, ist es für die Szenengraph-Kernentwickler schwierig, immer alle Möglichkeiten zu unterstützen. Vergleichbar zur oben genannten Erweiterbarkeit im Knotenbereich sollte es auch im Bereich der OpenGL-Erweiterungen Möglichkeiten geben, die der Anwendung erlauben, eigene Erweiterungen hinzuzufügen, ohne den Quellcode der Bibliothek ändern zu müssen.

OpenSG löst das Problem durch Gruppieren von OpenGL-Zuständen in größere Blöcke, sog. StateChunks. Dies ist motiviert durch die Beobachtung, dass manche OpenGL-Zustände praktisch immer zusammen mit anderen geändert werden,

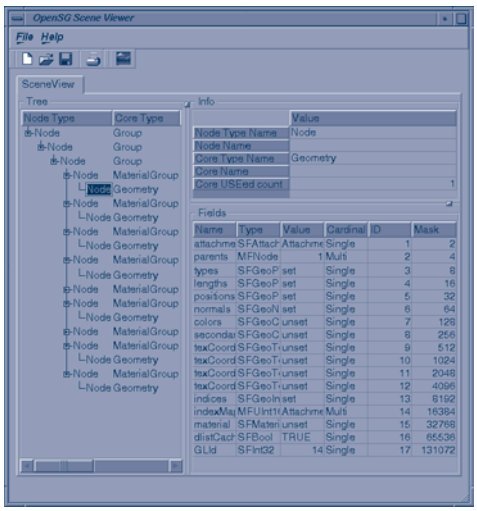

Abb. 4 Generisches User Interface zur Darstellung beliebiger OpenSG-Strukturen

z. B. werden üblicherweise alle Filterparameter für eine bestimmte Textur zusammengesetzt. Die Gruppierung in Chunks hilft, alle wichtigen Parameter zusammenzuhalten.

Darüber hinaus reduziert sie die Anzahl der zu verarbeitenden Zustandsvariablen. OpenGL hat einen sehr umfangreichen Zustand mit mehreren hundert Variablen. Eine zentrale Aufgabe des Szenengraphen ist die Minimierung von Zustandswechseln, da diese die oben erwähnten „pipeline flushes" und damit massive Performanzverluste hervorrufen können. In letzter Konsequenz ist dieser Vorgang auf das Traveling-Salesman-Problem abbildbar. Da er aber potentiell für jedes Bild, also bis zu 60-mal pro Sekunde, ausgeführt werden muss, ist es sehr von Vorteil, den Zustandsraum verkleinern zu können.

Der Zustand, der zur Darstellung eines bestimmten Objekts verwendet wird, wird durch einen Vektor dieser StateChunks repräsentiert, der dynamisch zur Laufzeit erweitert werden kann und somit in der Lage ist, anwendungsspezifische Erweiterungen aufzunehmen.

Ein Problem dieses Ansatzes ist, dass er sehr nahe an OpenGL liegt und vom Benutzer Detailwissen über OpenGL-Zustände und ihre Abhängigkeiten erfordert. Da es ein Ziel von Szenengraphen ist, den Benutzer von diesen Details zu entlasten, wird eine stärkere Abstraktion benötigt, die einfacher zu benutzen ist. Bei OpenSG ist dies durch das Materialkonzept realisiert. Materialien vermitteln zwischen einer für den Benutzer intuitiven Darstellung und den von OpenGL benötigten StateChunks.

Darüber hinaus kontrollieren sie über den Zustand hinausgehende Anforderungen zur Umsetzung von aktuellen Graphikalgorithmen.

Unterstützung aktueller Graphikalgorithmen

Die Entwicklung neuer Algorithmen zur realistischen Darstellung dreidimensionaler Szenen schreitet momentan sehr schnell voran, getrieben durch die weite Verbreitung von mächtiger Graphikhardware. Die Geschwindigkeit moderner Graphikhardware ist inzwischen so groß geworden, dass es möglich ist, ein Objekt mehrfach zu zeichnen. Die Bedeutung dieser Multipass-Algorithmen nimmt inzwischen schon wieder etwas ab, da aktuelle Hardware so flexibel programmierbar ist, dass viele Ansätze in einem Rendering-Vorgang realisiert werden können. Sie haben aber ihre Bedeutung, insbesondere bei der Darstellung von komplexen Materialien wie menschlicher Haut, noch lange nicht verloren. Eng verwandt mit den Multipass-Algorithmen sind Multistage-Algorithmen, bei denen mehrere unabhängige Teilbilder verwendet werden, um ein Ergebnisbild zu erhalten. Ein typisches Beispiel dafür ist die Schattenerzeugung mittels Shadow Buffers, bei der ein temporäres Bild aus dem Blickpunkt der Lichtquelle benutzt wird, um für jeden Punkt der Szene zu testen, ob er weiter als ein anderes Objekt von der Lichtquelle entfernt und somit im Schatten liegt.

All diese Ansätze sollten dem Benutzer eines Szenengraphen zur Verfügung stehen, ohne dass er über die Details nachdenken muss. Als Konsequenz reicht es nicht mehr aus, einen Rendering-Traversal zu verwenden, der direkt alle gefundenen Objekte zeichnet, sondern es muss eine Zwischenstruktur angelegt werden, die die zur Erzeugung des Bildes notwendigen Schritte aufzeichnet und später ausführt. Eine solche Zwischenstruktur, häufig Draw List oder Display List genannt, ist auch eine gute Stelle um die oben genannte Zustandswechselminimierung durchzuführen.

OpenSG erweitert das Prinzip, indem statt einer Liste ein Baum aufgebaut wird. Dieser Baum, der Draw Tree, kann sehr flexibel erzeugt und manipuliert werden und erlaubt dank seiner Baumstruktur auch sehr einfache Wiederverwertung von Teilszenen.

Die Struktur des Draw Tree ist von der des eigentlichen Szenengraphen getrennt. Theoretisch

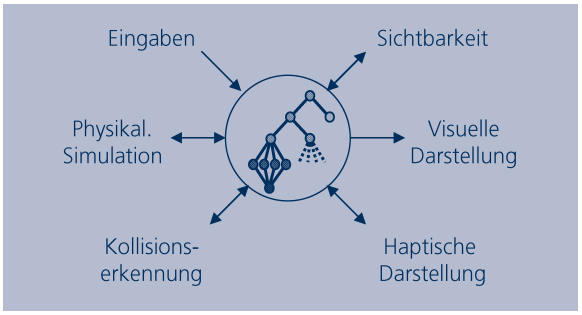

Abb. 5 Prozesse innerhalb eines VR-Systems, die Zugriff auf den Szenengraph benötigen

könnte auch ein temporärer, sortierter Szenengraph aufgebaut werden; manche Entwickler haben dies auch vorgeschlagen. Im Kontext moderner Systeme hat dies allerdings gravierende Nachteile, da die Knoten des Szenengraphen wesentlich komplexer sein können, als für den Draw Tree nötig ist. Sie sollten nämlich zum Beispiel Multithreading unterstützen.

Unterstützung von Multithreading

Multithreading, die Verwendung mehrerer unabhängiger Threads innerhalb einer Applikation, gewinnt immer mehr an Bedeutung. Zum einen durch die leichtere Verfügbarkeit von Mehrprozessorrechnern. So sind Doppelprozessorsysteme inzwischen in für jeden erschwingliche Preisbereiche gelangt. Zum anderen viel grundsätzlicher durch das Aufkommen von Multi-Core CPUs, bei denen mehrere Prozessorkerne gleichzeitig, unabhängig voneinander, verschiedene Programmcodes ausführen können. Um solche Systeme effizient nutzen zu können, muss auch der Szenengraph parallel arbeiten können und zugreifbar sein, denn er ist mehr als nur die Struktur zur Darstellung. In typischen VR-Systemen ist der Szenengraph der zentrale Datenspeicher für alle geometrischen Daten. Auf diese wird von einer Vielzahl Prozesse zugegriffen, teils nur lesend, teils aber auch schreibend (Abb. 5).

Für alle diese Prozesse ist es kritisch, dass sie auf einen konsistenten Szenengraph zugreifen können. Wenn andere Prozesse schreibend auf den Szenengraph zugreifen, während z. B. ein Simulationsschritt der physikalischen Simulation berechnet wird, werden die Ergebnisse unvorhersagbar und nutzlos.

Manche Systeme benutzen Locks, um Zugriffe auf einzelne Knoten zu synchronisieren, d. h. jeder Knoten beinhaltet ein Lock, das gesetzt werden muss, bevor der Knoten geändert werden darf. Dieses Vorgehen ist relativ aufwendig, da Locks in Multiprozessorsystemen zwischen allen Prozessoren synchronisiert werden müssen. Schlimmer noch, es löst das Problem nicht, da viele Aufgaben wie z. B. Simulationen viele verschiedene Knoten ändern müssen. Alle diese Knoten mit einzelnen Locks zu versehen, würde das Multithreading effektiv abschalten und den Nutzen mehrerer Prozessoren annulieren.

Prinzipiell benötigt jeder Thread eine komplette, private Kopie des Szenengraphs. Praktisch ist das nicht möglich, da herausfordernde Anwendungen schon mit einer Kopie den Speicher weitgehend ausnutzen. Es ist auch nicht wirklich nötig, da nicht jeder Thread wirklich alle Daten in jedem Schritt verändern muss. Daher sollte es möglich sein, große Teile der Daten gemeinsam zu benutzen und nur dann private Kopien anzufertigen, sobald Änderungen vorgenommen werden. Die Frage ist, welche Daten gemeinsam genutzt werden können.

Eine Untersuchung der Datenverteilung in typischen Szenen hat gezeigt, dass die Daten in zwei große Gruppen zerfallen. Auf der einen Seite die strukturellen Daten des Graphen, die Verbindungen zwischen Knoten und diejenigen Datenfelder, die nur ein einzelnes Element enthalten wie Objektnamen oder Farben von Materialien. Auf der anderen Seite die konstruktiven Geometriedaten wie Punkte und Normalen oder die Pixeldaten von Texturen. In den meisten Szenen ist das Verhältnis zwischen den beiden Datengruppen ungefähr 1:10. Daher wurde bei OpenSG entschieden, die Strukturdaten zu replizieren und die Konstruktivdaten gemeinsam zu verwenden.

Diese Details sollten den Anwendungsentwickler nicht belasten. Er sollte in der Lage sein, die Daten einfach zu benutzen, ohne an die Replikation und die Unterscheidung zwischen Struktur- und Konstruktivdaten denken zu müssen. OpenSG erreicht dies durch die Einführung eigener Pointertypen, die den Zugriff auf die Daten regeln [17].

Synchronisation zwischen den verschiedenen Kopien wird auf effiziente Weise ermöglicht, indem die geänderten Knoten und Felder in einer ChangeList protokolliert werden. Wenn ein Thread sich mit einem anderen synchronisieren will, muss er nur die ChangeList durchgehen und die geänderten Felder in seine Kopie überführen.

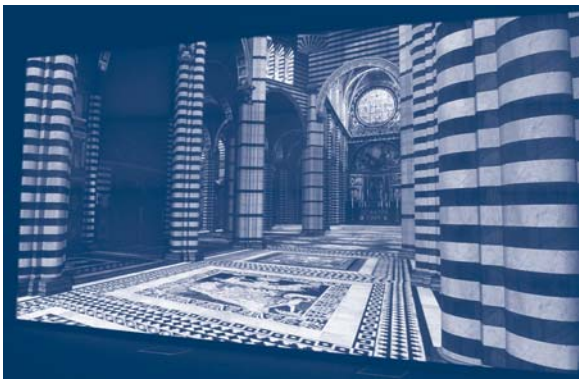

Abb. 6 48 Projektor/48 PC HeyeWall, 6000×3000 Pixel Auflösung. (Bild Fraunhofer IGD)

Unterstützung von Clustering

Während die Leistung einzelner handelsüblicher PCs die von Workstations inzwischen weit übertrifft, haben letztere in manchen Bereichen noch wesentliche Vorteile: sie können mehrere Graphikkarten aufnehmen. Dies ist wichtig im Zusammenhang mit aktuellen Trends im Bereich Ausgabegeräte für virtuelle Realität. Unter Verwendung von heute verfügbaren billigen digitalen Projektoren ist es möglich, Projektionswände mit extrem hohen Auflösungen aufzubauen. Ein Beispiel dafür ist die vor kurzen in Darmstadt eröffnete HeyeWall (Abb. 6), das unter Verwendung von 48 Projektoren ein Stereobild mit einer Auflösung von mehr als 6000×3000 Pixel zur Verfügung stellt.

Um ein solches System oder selbst kleinere Varianten davon zu betreiben, benötigt man eine entsprechende Anzahl Ausgabekanäle aus den Bildgeneratoren. Diese sind von Workstationherstellern zwar zu beziehen, die Kosten eines solchen Systems aber wären astronomisch. Die einzig praktikable Lösung ist die Verwendung von mehreren handelsüblichen Rechnern, die über entsprechende Software eine konsistente Applikation ausführen.

Es gibt verschiedene Möglichkeiten eine solche Applikation zu realisieren. Die Alternativen reichen von Verteilung auf OpenGL-Ebene [1], die keinerlei Veränderung an der Anwendung erfordert, aber viel Netzwerkverkehr erzeugt, bis hin zur Verteilung auf Applikationsebene, bei der die Anwendung die Verteilung von Daten und deren Synchronisation übernimmt, was eine drastisch angepasste Anwendung erfordert, aber mit minimalem Netzwerkverkehr einhergeht. OpenSG verwendet einen Mittelweg, bei dem die Änderungen des Szenengraphen übertragen werden. Diese stehen OpenSG direkt zur Verfügung, da sie für die Multithreadsynchronisation protokolliert werden. Diese Änderungen können leicht über ein Netzwerk übertragen werden, um auf der Gegenseite in einen lokalen Szenengraphen eingepflegt zu werden und somit eine vollständige Kopie des ursprünglichen Graphen zu erhalten. Dies ermöglicht die Verwendung von generischen Servern auf der Clusterseite, die alle von beliebigen Anwendungen getrieben werden, die ganz normal auf einem einzelnen Rechner laufen [17]. Einziger Unterschied und Anforderung an diese Anwendung ist, dass ein spezielles Fenster geöffnet werden muss, das die Daten über das Netzwerk verteilt und das notwendige Rendering synchronisiert. Üblicherweise dauert die Umstellung einer beliebigen OpenSG-Applikation auf Clusterbetrieb weniger als eine Stunde. Verglichen mit dem Aufwand eine Applikation auf Applikationsebene zu verteilen, ist dies vernachlässigbar.

Entwicklungsunterstützung

Open-Source-Systeme stehen neben den genannten technischen auch vor organisatorischen und finanziellen Herausforderungen. Ein wesentlicher Punkt dabei ist die Basisunterstützung. Um ein Open-Source-Projekt am Leben zu halten und um ihm eine konsistente Vision zu geben, benötigt es den konstanten Support eines Kernteams. Dieses Kernteam muss mehr als nur Freizeit zur Verfügung haben, um das Projekt voranzubringen, zumindest ein Teil ihrer Zeit sollte dediziert, und damit finanziert, zur Verfügung stehen. Darüber hinaus gibt es immer Aufgaben, die in einem Open-Source-Projekt nur selten freiwillig übernommen werden; ein typisches Beispiel dafür ist die Erstellung von Dokumentationsmaterial. Diese Aspekte lassen sich nur sinnvoll lösen, wenn dem Projekt ein Grundstock an Arbeitskapital zur Verfügung steht. OpenSG versucht dieses Problem mit Hilfe des OpenSG-Forums anzugehen.

OpenSG Forum

Das OpenSG-Forum ist ein Zusammenschluss von internationalen Firmen, Forschungsinstitutionen und Universitäten, die das OpenSG-Projekt als wichtig ansieht und durch ihre Mitgliedsbeiträge unterstützt. Darüber hinaus hilft es, dem Projekt mehr Gewicht zu verleihen und weitere Unterstützung zu gewinnen. Es dient auch als Kontrollgremium, um sicherzustellen, dass die geplanten Entwicklungen den Anforderungen der Anwender entsprechen und

Abb. 7 NURBS Rendering in OpenSG Plus. (Modell Volkswagen, Bild Uni Bonn)

sich nicht in rein experimentelle Gebiete verirren.

Die genaue Zusammensetzung ändert sich regelmäßig, sodass die aktuelle Mitgliederliste unter http://www.opensg.org/forum zu finden ist.

OpenSG Plus

Das OpenSG-Forum sichert die Basisfinanzierung von OpenSG, aber es kann nicht die Entwicklung abdecken. Um weitere Expertisen zu erhalten und um mehr Interesse bei Universitäten zu wecken, wurde im Rahmen des BMBF VR/AR Ideenskizzen Programms das Projekt OpenSG Plus beantragt und zur Förderung angenommen. Im Rahmen von OpenSG Plus wurden eine Reihe von Universitäten (RWTH Aachen, TU Braunschweig, Uni Bonn, Uni Stuttgart, Uni Tübingen) und außeruniversitäre Forschungseinrichtungen (MPI Saarbrücken, Fraunhofer IGD, ZGDV Darmstadt) unterstützt, um neue Forschung in ihrem Spezialgebiet auf OpenSG aufzusetzen. Die behandelten Themen waren sehr vielfältig und gliedern sich in 4 große Bereiche:

- Kern,
- komplexe Szenen,
- komplexe Primitive und
- komplexe Beleuchtung.

Teile der Kernentwicklung, wie die Reflektivität, die State- und Materialabstraktion oder das Clustering, wurden durch OpenSG Plus ermöglicht. Darüber hinaus wurden Arbeiten zu geometrischen Datenstrukturen (OpenMesh) durchgeführt. Im Bereich komplexe Szenen lag der Focus auf effizienter Hierarchiegenerierung und der Ausnutzung der Hierarchie für Occlusion Culling. Komplexe Primitive umfasste geometrische Primitive wie NURBS (Abb. 7) und Subdivison Surfaces, aber auch nichtgeometrische Primitive wie Volumina (Abb. 8). Komplexe Beleuchtung schließlich umfasste sowohl die Entwicklung verschiedener Beleuchtungsalgorithmen wie NDF Shading [4] als auch die Konzeptionierung eines allgemeinen Integrationskonzepts für Shaderintegration in Szenengraphen.

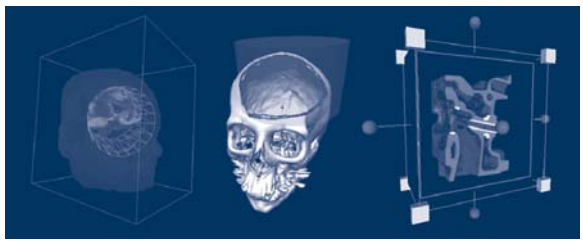

Abb. 8 Volumenrendering in OpenSG Plus. (Bild Uni Stuttgart)

Die Ergebnisse des Projekts wurden unter anderem auf zwei OpenSG-Symposien präsentiert (Publikationen unter [11,12]). Das Projekt wurde im Dezember 2003 beendet, und die Ergebnisse werden sukzessive in den OpenSG-Kern integriert.

Zusammenfassung

3D-Graphikhardware ist inzwischen überall leicht verfügbar, was auf der einen Seite viele neue Entwickler in die Forschung und Entwicklung von 3D-Darstellungsalgorithmen zieht, auf der anderen Seite viele neue Anwender, die nicht unbedingt tief in die Forschung einsteigen, sondern nur ihre Problem lösen wollen. Szenengraphen sind ein gutes konzeptionelles Mittel, um zwischen diesen beiden Welten zu vermitteln, da sie in der Lage sind, die Komplexität der unterliegenden 3D-Hardware und -Algorithmen vor dem Benutzer verstecken zu können. Szenengraphen sind recht weit verbreitet und es existieren einige kommerzielle wie auch Open-Source-Implementierungen, wobei die letzteren häufig besser sind.

Die heutigen Systemumgebungen und -anwendungen stellen einige neue Anforderungen an Szenengraphsysteme, wie weitgehende Erweiterbarkeit, zur Unterstützung neuer Graphikhardwarefeatures, -algorithmen sowie die Unterstützung von Multithreading und Clustering. Dieser Beitrag zeigt exemplarisch die Ansätze, die OpenSG verwendet, um diese Probleme zu lösen.

OpenSG ist als Open-Source-System unter http://www.opensg.org frei verfügbar, und wird auch in Zukunft weiterentwickelt. Offene Fragen

sind die nahtlose Integration von Programmierbarkeitsfeatures aktueller Hardware, die mehr auf hart programmierte Effekte in Spielen ausgelegt sind, als auf die freie Zusammenstellbarkeit eines Szenengraphen. Weitere ständig aktuelle Themen sind Dokumentation und Performance-Verbesserung.

Nichtsdestotrotz erfüllt es alle genannten Anforderungen und wird in vielen Anwendungen in verschiedenen Universitäten und Forschungseinrichtungen erfolgreich eingesetzt.

Literatur

1. Chromium:http://chromium.sourceforge.net/
2. Coin:http://www.coin3d.org
3. Gamma, E.; Helm, R.; Johnson, R.E.; Vlissides, J.: Design patterns: Abstraction and reuse of object-oriented design. In: ECOOP ‚93 – Object-Oriented Programming 7th European Conference, Germany, July 1993. Proceedings. Lecture notes in Computer Science. Berlin Heidelberg New York Tokio: Springer 1993
4. Kautz, J.; Daubert, K.: User-defined shading models for VR applications, In OpenSG Symposium 2002,http://www.opensg.org/OpenSGPLUS/symposium/Papers2002/Kautz_NDF.pdf
5. Microsoft: DirectX 9/Direct3D:http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000410
6. OpenGL Architecture Review Board: The OpenGL Specification, Version 1.5.http://www.opengl.org/developers/documentation/specs.html
7. Open Inventor:http://oss.sgi.com/projects/inventor/
8. OpenRM.http://www.openrm.org
9. Open Scenegraph:http://www.openscenegraph.org
10. OpenSG:http://www.opensg.org
11. OpenSG Symposium 2002 Publikationen:http://www.opensg.org/OpenSGPLUS/symposium/Papers2002/index.DE.html
12. OpenSG Symposium 2003 Publikationen:http://www.eg.org/EG/DL/PE/OPENSG03
13. Rohlf, J.; Helman, J.: IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. Proceedings of SIGGRAPH 94
14. SGL:http://sgl.sourceforge.net/
15. Strauss, P.; Carey, R.: An object-oriented 3D graphics toolkit . Proceedings of SIGGRAPH 92
16. Sun Microsystems: Java3D API,http://java.sun.com/products/java-media/3D
17. Voss, G.; Roth, M.; Reiners, D.; Behr, J.: A Multi-thread safe foundation for scenegraphs and its extension to clusters. Fourth Eurographics Workshop on Parallel Graphics and Visualization, 2002