Handbuch zur Visualisierung am ZIB

Abt. Visualisierung und Paralleles Rechnen

Zusammenfassung

Scientists intending to employ scientific visualization techniques are confronted with the unpleasant task of choosing visualization software that meets their specific needs. A good choice requires consideration of several aspects. Most important are functionality, i.e. the set of supported visual representations, and restrictions due to the user's production environment (i.e. available hardware and software platforms, software compatibility constraints, need of specific input and output formats, network distribution requirements). Crucial are quality demands, ranging from simple graphics for interactive control of simulations, to images of highest quality (glossy prints, slides or video movies) for presentation purposes. Further aspects of consideration are: ease of use, level on which the user is willing to program, portability, conformity to prevalent standards, availability in the world-wide user community, future dissemination, as well as emerging trends in computer graphics and scientific visualization.

The aim of this manual is to give a survey on the graphics and visualization software that is currently being provided at ZIB (mainly for internal use). The whole range of computer graphics software relevant in mathematical, natural and technical sciences is covered: image synthesis (basic and higher graphics libraries, plot programs, visualization environments, combined text and drawing programs, renderers), image processing and storage (raster and vector formats, page description languages, raster toolkits for image processing and format conversion), capturing of images as well as printing on paper and recording on photographic film or video tape.

This information together with a glossary and recommendations for further reading should help prospective users in getting started and assist them in making good choices of graphics and visualization software.

Vorwort

Die *wissenschaftliche Visualisierung* (Scientific Visualization) ist ein relativ junges, aber rasant wachsendes Gebiet, in dem Resultate aus einer Reihe informatischer Teilgebiete mit denen vieler anderer wissenschaftlicher und technischer Disziplinen zusammenfließen. Ihr Ziel ist, dem Anwender die Interpretation großer und komplexer Datenmengen mittels computerberechneter Bilder zu erleichtern oder gar erst zu ermöglichen. Die zu visualisierenden Daten können Meßwerte sein oder aus Computersimulationen stammen.

Die Visualisierung basiert auf den hochentwickelten visuellen Wahrnehmungsfähigkeiten des Menschen, insbesondere auf der Fähigkeit, Strukturen und Trends zu identifizieren, Muster und Formen zu erkennen oder Unwesentliches auszublenden. Sie kann zu einem intuitiven Verständnis von komplexen Sachverhalten führen, die durch Sichtung von Zahlenkolonnen oder einfachen Kurvendiagrammen kaum erfaßbar wären.

Die Lösung umfangreicher Probleme mit den Methoden des *Scientific Computing*, wie auch die Entwicklung moderner Simulationsmethoden selbst, ist zunehmend von der Verfügbarkeit leistungsfähiger Visualisierungsmethoden abhängig. Dies führte in den USA Ende der Achtziger Jahre zu der breit angelegten und von der NSF großzügig geförderten Initiative „Visualization in Scientific Computing“ [39]. Dadurch sind in den USA, später auch weltweit, an vielen Supercomputer-Zentren und Forschungsinstituten Arbeitsgruppen entstanden, die sich ausschließlich der Forschung, Entwicklung und Produktion im Bereich der Visualisierung widmen. Es wurde Software entwickelt, die es auch Nichtexperten erlaubt, aufwendige Visualisierungstechniken zu nutzen, so daß komplexere Visualisierungsaufgaben von den Fachwissenschaftlern selbst gelöst werden können. Für die Produktion von Einzelbildern oder Filmen hoher Qualität, etwa für Präsentationszwecke, ist jedoch meist doch ein Team von Visualisierungsexperten erforderlich, trotz aller Bemühungen, einfach bedienbare Visualisierungssoftware zu schaffen. „Produktion“ ist daher, neben Entwicklungs- und Forschungsaufgaben, eine wichtige Aufgabe der in Visualisierungszentren tätigen Arbeitsgruppen.

Dementsprechend sieht die im ZIB tätige Visualisierungsgruppe ihre Aufgaben in der Bereitstellung von Computergrafik- und Visualisierungssoftware, der Unterstützung hausinterner Anwender bei der Realisierung komplexer Visualisierungsprojekte, der Entwicklung von Visualisierungsmethoden entsprechend den Anforderungen aus der wissenschaftlichen Arbeit des ZIB und zunehmend auch der Forschung im Bereich der wissenschaftlichen Visualisierung.

Die Bereitstellung von Software ist wegen der ständigen Weiter- und Neuentwicklungen eine permanente Aufgabe. Mit diesem Handbuch wird eine Momentaufnahme der Software-Situation im Visualisierungsbereich am ZIB gegeben.

Angesichts des bloßen Umfangs von Visualisierungssystemen und der zugehörigen Dokumentation ist es offensichtlich, daß der suchende Anwender sich nicht probeweise in ein zufällig erreichbares System vertiefen sollte. Bevor er

sich Informationen über ein spezifisches Visualisierungspaket besorgt, benötigt er einen - zumindest teilweisen - Überblick über die insgesamt angebotene Software. Leider ist das Konglomerat an weltweit verfügbarer Visualisierungssoftware nirgendwo dokumentiert.

Aus dieser Situation heraus entschlossen wir uns, trotz der ständigen, teils rasanten Veränderungen, den Anwendern im ZIB mit einem Handbuch einen Überblick über eine Auswahl der am ZIB bereitgestellten Grafik- und Visualisierungssoftware zu geben.[1] Hauptanliegen dieses Handbuchs ist es, dem wissenschaftlichen Anwender, ohne ihn zum Experten auszubilden, eine schnelle Informationsmöglichkeit und einen knappen Überblick über verfügbare Software zu verschaffen. Es soll ihn in die Lage versetzen, das für seine Anwendungen und seine Produktionsumgebung geeignete Grafik- oder Visualisierungswerkzeug schnell ausfindig zu machen. Das Handbuch ist knapp gehalten und im wesentlichen auf eine Übersicht beschränkt. Etwas weiter ausgeführt werden nur Themen, die nirgendwo in der einschlägigen Literatur zusammenhängend behandelt sind, wie etwa die Konversion von Rasterbild-Formaten. Auf die Darstellung anderswo gut erklärter Sachverhalte wurde verzichtet. Insbesondere bietet das Handbuch keinen Ersatz für Originaldokumentationen, deren Umfang, selbst für ein einzelnes Software-Produkt, den des Handbuches oft um Größenordnungen übersteigt. Zur Erleichterung der Lektüre weiterführender Computergrafik- und Visualisierungsliteratur wurde ein Glossar angefügt (Anhang B), das eine Klärung der wichtigsten Fachbegriffe enthält. Das Handbuch bietet *keine* Einführung in Visualisierungstechniken[2] und wird den Anwender nicht befähigen, selbstständig komplexere Produktionen, wie etwa die Erstellung eines ästhetisch ansprechenden Videofilms, durchzuführen. Für die Lösung solcher Aufgaben bietet die Visualisierungsgruppe des ZIB praktische Unterstützung.

In der ersten Auflage des Handbuchs stehen die herkömmlichen Grafikpakete im Vordergrund; für die nächste Auflage ist eine Verlagerung des Schwerpunkts auf fortgeschrittene Visualisierungssysteme beabsichtigt.

Jede Anregung für zukünftige Auflagen des Handbuches, wie auch zur Verbesserung der Software-Situation im Bereich Visualisierung am ZIB, wird dankbar aufgenommen.

Ich möchte an dieser Stelle allen Mitarbeiter danken, die durch Anregungen, Ratschläge, kleine oder umfassende Beiträge am Zustandekommen dieses Handbuches beteiligt waren. Hierzu zählen T. Brenner, H.-H. Frese, T. Höllerer, H. Jaedicke, J. Langendorf, O. Paetsch, K. Peter, G. Skorobohatyj, D. Stalling, Dr. T. Steinke und R. Wunderling.

Hans-Christian Hege

[1] Wegen der im ZIB getroffenen Festlegung auf UNIX wurde *nur UNIX-Software berücksichtigt. Weiterhin erhielt die Darstellung von Software für Silicon-Graphics-Rechner etwas mehr Gewicht als die anderer Rechnertypen, da im ZIB aus Hard- und Software-Gründen vorzugsweise Rechner dieses Typs für Visualisierungsaufgaben eingesetzt werden.*

[2] Als weiterführende Literatur kann empfohlen werden: zum Thema Computergrafik: [19, 23, 5, 24, 53, 68, 13], zum Thema Visualisierung: [39, 12, 8, 37], zum Thema Farbe: [27, 10, 1] und zum Thema Video: [69, 67].

Inhaltsverzeichnis

Excellence in [statistical] graphics consists of complex ideas communicated with clarity, precision, and efficiency. General displays should

- *show the data*
- *induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production, or something else*
- *avoid distorting what the data have to say*
- *present many numbers in a small space*
- *make large data sets coherent*
- *encourage the eye to compare different pieces of data*
- *reveal the data at several levels of detail, from a broad overview to the fine structure*
- *serve a reasonably clear purpose: description, exploration, tabulation, or decoration*
- *be closely integrated with the statistical and verbal description of a data set.*

Graphics reveal *data.*

– Edward R. Tufte, *The Visual Display of Quantitative Information*, 1983, Cheshire

Kapitel 1

Visualisierungssoftware

Die Grafik- und Visualisierungsbedürfnisse der Anwender im Scientific Computing sind vielfältiger Art. Sie reichen von grundlegenden und höheren Grafik-Funktionsbibliotheken, über fertige Plot- und sonstige Grafikprogramme, bis hin zu flexiblen, aber aufwendigen Visualisierungsumgebungen und High-End-Animationssystemen. Leider existiert kein universelles Visualisierungssystem, mit dem sich alle Visualisierungsaufgaben, vom einfachen xy-Plot bis hin etwa zum animierten, applikationsspezifischen Volume-Rendering, abdecken lassen.

Bei der Auswahl von Visualisierungssoftware ist zu beachten, daß der rein grafisch-funktionale Aspekt nur eine von vielen Dimensionen ist. Daneben sind weitere Aspekt zu berücksichtigen und gegeneinander abzuwägen:

- Normkonformität bzw. Standardnähe, Portabilität und Verbreitung $\leftrightarrow$ (meist) Modernität
- Unterstützung unterschiedlicher Arbeitsweisen wie etwa
  - netzverteilte $\leftrightarrow$ unverteilte
  - interaktive $\leftrightarrow$ programmgesteuerte

  - Anwendungen und Visualisierung direkt gekoppelt ↔ Post-Processing

- Eignung für die jeweilige Zielgruppe, z.B.
  - erfahrene Grafikprogrammierer
  - Anwendungsprogrammierer
  - (eventuell programmierunwillige) Endanwender
- Komplexität der Bedienung, erforderliche Einarbeitungszeit
- Universalität, Generizität ↔ Zuschnitt auf spezifische Anwendungen
- Berücksichtigung der (meist weitgehend) vorgegebenen Produktionsumgebung
  - Hardware-Voraussetzungen[1]
  - Software-Voraussetzungen (Betriebssystem, Basis-Bibliotheken, usw.)
  - Kompatibilität mit anderen Programmen
  - gewünschte Ein- und Ausgabe-Formate
- Qualitätsanforderungen (minimal für interaktive Kontrolle von Simulationen und maximal für „Präsentationszwecke“)
- Geschwindigkeitsanforderungen (etwa maximale für interaktive Kontrolle, und minimale für die Vorbereitung von Präsentationen)
- Zukunftsträchtigkeit (wichtig bei längerfristigen Projekten).

Eine gute Software-Lösung für ein vorgegebenes Visualisierungsproblem zu finden, setzt voraus, daß man sich in diesem mehrdimensionalen Merkmalsraum zurechtfindet. Die Vielzahl der relevanten Aspekte erklärt, weshalb eine Nutzerschaft mit unterschiedlichen Anwendungen in einem fachlich breit angelegten Institut nicht durch ein einziges leistungsfähiges Visualisierungsprogramm, sondern nur durch eine ganze Reihe von Programmsystemen zufrieden gestellt werden kann, deren grafische Funktionalität sich vielfach überscheiden kann.

Wegen der raschen Entwicklung von Visualisierungstechniken, -algorithmen und den generellen Fortschritten in der Software-Technik sowie der rasanten Hardware-Entwicklung (etwa im Bereich Spezialhardware für Grafik und Multimedia) ändert sich die Software-Situation ständig. Ein Visualisierungssystem zu entwickeln, das Chancen auf weite Verbreitung in der akademischen Welt hat, ist heute, selbst bei kostenfreier Verteilung des Produkts, ein aufwendiges Unterfangen. Die letzte erfolgreiche nicht-kommerzielle Initiative dieser Art, das Khoros-Projekt der University of New Mexico, erforderte die Bindung enormer Ressourcen und war nur in weltweiter Zusammenarbeit mit anderen, auf Bildsynthese und -verarbeitung spezialisierten wissenschaftlichen Instituten möglich. Selbst große kommerzielle Anbieter sehen sich nicht mehr in der

[1] Grafiksoftware ist meist enger an spezifische Hardware gebunden als andere Anwendungssoftware.

Lage, die Vielzahl der Visualisierungsbedürfnisse zu decken. Daher findet die Idee, leistungsfähige, *erweiterbare Grund*systeme zu etablieren und dann die weltweiten Entwicklungsressourcen der auf verschiedenste Anwendungsgebiete spezialisierten Wissenschaftler zu nutzen, immer mehr Anklang. Die Tatsache, daß im Public-Domain-Bereich ein ziemlicher Wildwuchs herrscht, tut dieser Idee keinen Abbruch. Im Laufe der Zeit wird sich das für die Anwendungen am besten Geeignete durchsetzen.

Mit modernen, erweiterbaren, modul- oder objektorientierten Visualisierungssystemen wurden vor wenigen Jahren noch undenkbare, teils auch unvereinbar erscheinende Qualitäten Wirklichkeit. Diese Systeme sind jedoch nur potentiell mächtig: dem Anwender stehen hauptsächlich Grundfunktionen zur Verfügung. Die Kerne der einzelnen Module muß er selbst programmieren oder aber – ein immer wichtiger werdender Gesichtspunkt – sich aus den in weltweiter Zusammenarbeit entstehenden Software-Pools besorgen. Das Aufkommen solcher Systeme ändert die Arbeitsweisen. Andere Fragen treten in den Vordergrund, etwa:

- Wie erhält man Informationen über die riesige Palette weltweit verfügbarer Software ?
- Wie wählt man für den jeweiligen Anwendungsfall geeignete Software aus?
- Wie löst man eine gegebene Aufgabe unter (teilweiser) Nutzung verfügbarer Module möglichst effizient ?
- Welche der Software-Pakete kann man kombinieren, wie geschieht das ?
- Wie erweitert man ein gegebenes Software-System ?

Das vorrangige Ziel der Visualisierungsgruppe am ZIB im Hinblick auf die Bereitstellung von Software ist, möglichst universelle Werkzeuge anzubieten und davon nur so viele wie nötig zum Abdecken des gesamten Spektrums. Universalität bedeutet für den Anwender meist aber aufwendige Vorbereitung und Bedienung. Daher ist diese Software durch geeignete *anwendungsspezifische* Software zu ergänzen. Für beide Ebenen nehmen wir Eigenentwicklungen vor oder bedienen uns kostenlos verfügbarer sowie kommerzieller Fremdsoftware.

Um die „Interoperability“ der verschiedenen Pakete ist es derzeit noch schlecht bestellt: Brücken sind meist nur durch Eigenentwicklungen zu schlagen.

One plus one equals three or more: factual facts and actual facts.

– Josef Albers, *Search versus Re-Search*, 1969, Hartford

Kapitel 2

Erzeugung von Bildern

2.1 Funktionsbibliotheken

Grafische Software macht vielfach Gebrauch von einigen, immer wieder verwendeten Grundfunktionen. Es liegt daher nahe, diese dem Anwendungsprogrammierer in Form von vorgefertigten Funktionsbibliotheken zur Verfügung zu stellen. Inzwischen gibt es eine ganze Reihe von Basis-Grafik-Funktionsbibliotheken. Die wichtigsten sind X und GKS für 2-D-Grafik sowie die beiden Familien PHIGS/PHIGS-PLUS/PEX und GL/DGL/OpenGL für 3-D-Grafik.

Eine Reihe von Funktionsbibliotheken, insbesondere die der Rechnerhersteller, sind an spezifische Hardware gebunden. Doch auch Grafiksoftware sollte portabel sein, sich also ohne großen Aufwand an möglichst viele Geräte und Systemkonfigurationen anpassen lassen. Dazu muß sie möglichst unabhängig sein von den verschiedenen Ein- und Ausgabegeräten (Bildschirmen, Druckern, Plottern, Mäusen, usw.) und speziellen Grafikkarten (Grafikbeschleunigern). Zur Entwicklung derart flexibler Software wurden *Standards* definiert. Sie erlauben eine weitestgehend maschinenunabhängige Programmierung. Die Standards bestehen aus Funktionsbibliotheken mit Funktionen, auf deren Existenz und definierte Eigenschaften sich der Anwendungsprogrammierer verlassen kann. Internationale (ISO) und nationale (ANSI, DIN) Standards sind heute die Bibliotheken GKS und PHIGS. Eine Weiterentwicklung von PHIGS, das PHIGS-PLUS, ist zum Teil als Standard verabschiedet. Die weiteste Verbreitung hat X erlangt; es ist zwar nicht genormt (das würde seine Weiterentwicklung nur bremsen), aber dadurch, daß seine Entwicklung von nur einer Institution kontrolliert wird, innerhalb der jeweiligen Entwicklungsstufe einheitlich.

Die Kehrseiten der Nutzung von Standards sind folgende: Standardisierung ist ein kompliziertes, sich über mehrere Jahre erstreckendes Unterfangen, so daß Standards zum Zeitpunkt ihrer Verabschiedung zwangsläufig nicht mehr "state-of-the-art" sind. Darüber hinaus erzwingt die Geräteunabhängigkeit einen gewissen Overhead und verhindert eine effiziente Nutzung von Spezialhardware. Interaktive Grafik, also die Berechnung von mehreren Millionen Bildpunkten pro Sekunde, erfordert insbesondere im 3-D-Bereich Rechenleistungen, die man (derzeit) nur durch spezialisierte parallele Grafikhardware erreichen kann. De-

ren Leistungsfähigkeit voll auszuschöpfen setzt eine enge Anpassung der unteren Software-Schichten an die Hardware voraus. Paradebeispiel für eine leistungsfähige Hard- und Software-Kombination ist die Grafik-Hardware und GL-Bibliothek von Silicon-Graphics (SGI). Die für lange Zeit einzigartige Leistungsfähigkeit dieser Kombination sorgte dafür, daß die für viele Anwendungsgebiete beste Grafik- und Visualisierungs-Software heute typischerweise auf der GL basiert.

Natürlich läßt sich die GL-Bibliothek auch an andere Hardware anpassen. In einigen Fällen ist dies auch geschehen. Um eine noch weitere Verbreitung GL-basierter (also SGI-komptibler) Software zu erreichen und diese von Hardware, Betriebs- und Windowsystem unabhängig zu machen, versucht SGI in jüngster Zeit, mit der OpenGL einen neuen „Industriestandard“ zu lancieren. Diese Entwicklung steht in direkter Konkurrenz zu PEX, einer Erweiterung des X auf 3-D-Grafik (mit PHIGS-PLUS-Funktionalität). Die Erfolgschancen von OpenGL und PEX werden sich frühestens Ende 1993 bemessen lassen.
Empfehlungen: Verwenden Sie, bis auf die GL, keine herstellerabhängige Bibliothek. Für 2-D-Grafik empfiehlt sich derzeit X. Im Bereich der 3-D-Grafik ist die Entscheidung schwieriger: es läßt sich derzeit schwer vorhersagen, ob sich PEX, OpenGL oder beide am Markt durchsetzen werden. Für PEX spricht, daß es gegenwärtig schon verfügbar ist.[1] Für die OpenGL spricht der bessere Zuschnitt auf die Möglichkeiten heutiger Grafik-Hardware (z.B. Texture Mapping) und die bessere Performance. Das Argument der plattform-neutralen Netzwerk-Fähigkeit spricht zunächst für PEX, könnte aber mit der Entwicklung der OpenGL Extension (GLX), die eine Integration der OpenGL in X zum Ziel hat, bald hinfällig werden.

Neben den Basisbibliotheken stehen dem Entwickler auch Bibliotheken mit höheren Grafikfunktionen zur Verfügung. Hier gibt es allerdings noch weniger Einheitlichkeit: zu allen Basisbibliotheken existieren Sammlungen mit höheren Funktionen. Beispielhaft seien hier nur je zwei Bibliotheken für den 2-D- und den 3-D-Bereich genannt: die MiniGraphik und die Xgraphic für 2-D, sowie die NAG-Graphics-Library als Vertreter herkömmlicher Funktionsbibliotheken und die weitaus modernere, objektorientierte Klassenbibliothek Inventor für 3-D. Wesentliche Auswahlkriterien für höhere Bibliotheken sind deren Funktionalität und die zugrundegelegte Basisbibliothek.

Ein sehr weiter Bereich, der hier *nicht* ausführlich zur Sprache kommen kann, ist die Erstellung von Graphical User Interfaces (GUIs) unter Verwendung spezieller höherer Funktionsbibliotheken (Widget Sets) und GUI-Builder. Hierzu verweisen wir auf die im Abschnitt X-Toolkits genannten Referenzen.

2.1.1 X

Das X Window System (kurz: X) stellt eine von Hardware und Betriebssystem unabhängige Schnittstelle zur Ansteuerung von monochromen (zwei und mehr Graustufen) und mehrfarbigen Raster-Displays zur Verfügung.

[1] Vorsicht: für ernsthafte, größere Entwicklungen nicht die im X11 Release 5 enthaltene PEX-Implementierung, sondern eine der von kommerziellen Anbietern erhältlichen solideren Implementierungen verwenden.

Es unterstützt neben einer hierarchischen Fensterverwaltung auch 2-D-Grafik- und Text-Operationen. Somit bildet es die Grundlage für grafische Benutzungsoberflächen (Graphical User Interfaces - GUI), die mit Hilfe von auf X aufbauenden Bibliotheken recht komfortabel erstellt werden können (s. *X-Toolkits*).

Das X-Window System besteht eigentlich nur aus einem Protokoll, dem *X-Protokoll*. Dieses regelt die Kommunikation zwischen einem *X-Server*, dem X-Programm, das Bildschirm, Tastatur und Maus ansteuert, also eine Abstraktion von Hardware leistet, und den auf X basierenden Anwendungsprogrammen, den sogenannten *X-Clients*.

Im Gegensatz zu vielen anderen Window-Systemen baut X auf einem Netzwerk-Protokoll auf und nicht auf einer nur betriebssystemspezifischen Schnittstelle. Eine Applikation kann so auf einem Rechner laufen und ihre Ausgaben in ein Fenster auf einem anderen Rechner mit einer eventuell ganz anderen Architektur ausgeben.

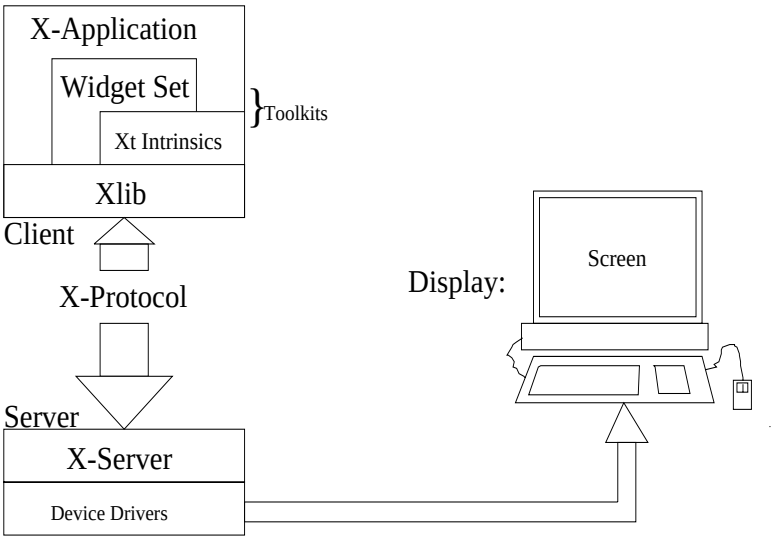

Abbildung 2.1: X Window System: Konzeption

Die Funktionsbibliothek *Xlib* ist die Basisbibliothek des X-Window Systems. Sie stellt dem Programmierer verschiedene Low-Level-Funktionen auf der Ebene des X-Protokolls zur Verfügung. Diese Ebene ist äußerst flexibel, erfordert allerdings einen erheblichen Programmieraufwand. Im Normalfall wird man als Programmierer von X-Applikationen nur auf eine stark begrenzte Auswahl von Xlib-Funktionen zurückgreifen und ansonsten mächtigere Funktionen von übergeordneten Bibliotheken benutzen (s. *X-Toolkits*). Erste Anlaufstelle für Informationen, die das tägliche Arbeiten mit dem X-Window System betreffen, wie z.B. Standard-Übergabe-Parameter für X-Applikationen, Ressourcen im X-Environment [2] oder auch über Fonts in X, ist die Manual Page zu X.

Als weiterführende Literatur sei auf die bei O'Reilly & Associates, Inc. erschie-

[2]Hierüber lassen sich bestimmte Voreinstellungen für das Erscheinungsbild von Applikationen setzen, wie z.B. Farben der Window-Elemente.

nene Bücherserie zum X Window System verwiesen: zum X-Protokoll s. [56], zur Xlib (Low-Level-Interface) s. [44] und zu diversen X-Applikationen s. [46].

X-Toolkits

Typische Applikationen mit grafischen Benutzungsoberflächen werden mit Hilfe von *X-Toolkits* aufgebaut.

Unter einem X-Toolkit versteht man die Bibliothek *Xt* (X Toolkit Intrinsics) und ein *Widget-Set*. Die X-Toolkit-Intrinsics sind eine mit Hilfe von Xlib-Funktionen implementierte Bibliothek. Sie vereinfachen den Zugang zu X beim Programmieren erheblich, z.B. durch Übernahme der Eventverwaltung. Xt dient unter anderem auch zur Implementierung von Widgets.

Widgets (*wi*ndow ga*dget*s) sind vorgefertigte graphische Grundobjekte einer Benutzungsoberfläche, die mit gewisser Funktionalität versehen sind, z.B. Knöpfe (buttons), die per Mausklick automatisch hervorgehoben werden und ein vom Programmierer definiertes Unterprogramm aufrufen. Neben solchen sichtbaren Standardelementen von X-Fenstern gibt es übergeordnete Widgets, die zur Anordnung und Verwaltung von Widgets in Gruppen dienen. All diese Widgets werden dem Programmierer in Funktionsbibliotheken, eben den *Widget-Sets*, zur Verfügung gestellt.

Ein Widget-Set besteht aus *Klassen* im Sinne der objektorientierten Programmierung. Die Widgetklassen sind ihrer Funktionalität nach hierarchisch angeordnet. Innerhalb der Hierarchie wird Funktionalität vererbt.

Es gibt verschiedene Widget-Sets. Die verbreitesten sind *Motif* von OSF und *OpenLook* von AT&T. In der Auslieferung von X (MIT-Version) ist das *Athena Widget Set*, Xaw enthalten, ein kleiner Satz von Widgets, mit dem einfache Applikationen geschrieben werden können.

Die drei erwähnten Widget-Sets sind alle in C, aber in objektorientiertem Stil implementiert. Sowohl das *OpenLook*- wie auch das *Motif*-Widget-Set werden mit einem Style Guide ausgeliefert, der Design-Vorschläge für ein einheitliches Aussehen der mit diesen Widgets programmierten Applikationen gibt. Der Anwender hat durch die Verwendung eines einheitlichen Widget-Sets in verschiedenen Anwendungen den Vorteil, daß die Bedienelemente der Anwendungen ein aufeinander abgestimmtes Aussehen und eine einheitliche Funktionalität besitzen. Da die Anwendungen letztendlich aber alle auf das X-Protokoll aufsetzen, sind sie in jeder Bedienoberfläche, die auf dem X-Window-System basiert, ablauffähig — es sei denn, sie wurden mit *shared libraries* gebunden, die dann natürlich bei Programmstart vorhanden sein müssen.

Literatur zu Intrinsics: s. [43, 75], zu Motif s. [45, 42].

2.1.2 MiniGraphik

MiniGraphik ist eine als Quellcode verfügbare Bibliothek von Basis-Grafik-Routinen, die eine einheitliche Schnittstelle für die Ausgabe von einfachen(!) 2-D-Grafiken auf verschiedenen Grafikumgebungen (X11, Mac, auch SunView) anbietet. Die Ausgabe erfolgt in Bildschirmfenster oder PostScript-Dateien, die auf einheitliche Weise als Ausgabemedium geöffnet werden. Außerdem ist auch

ein „Mitschnitt“ von Grafikausgaben eines Fensters in eine PostScript-Datei mit sehr geringem Aufwand möglich.

Die Bibliothek bietet neben den Basisroutinen zur Grafik- und Textausgabe auch Routinen an, die in Bildschirmfenstern Interaktion des Benutzers mit der *MiniGraphik* ermöglichen.

Die Programmierschnittstelle ist sauber und unkompliziert, so daß dem Anwender für immer wiederkehrende einfache grafische Anforderungen in seinen Programmierarbeiten eine maschinenunabhängige Schnittstelle zur Verfügung gestellt wird. Dies gewährleistet (wenigstens von der grafischen Seite) eine hohe Portabilität.

Die Bibliothek entstand in der Abt. Numerische Software-Entwicklung des ZIB, motiviert durch immer wieder anfallende einfache Grafikausgaben. *MiniGraphik* hat nicht den Anspruch eines umfangreichen Grafiksystems, sondern will ein praktischer Helfer für klar umrissene Problemstellungen sein (und ist es).

2.1.3 Xgraphic

Xgraphic ist ein weiteres System von einigen als Quellcode verfügbaren Funktionen zur Realisierung immer wiederkehrender einfacher Window-Operationen. Somit wird das Erstellen von (allerdings sehr stark standardisierten) grafischen Benutzungsoberflächen unterstützt.

Objekte wie Slider, Buttons, Textfelder lassen sich in einem Fenster vordefinierten Typs, dem *panel window* realisieren. Weiterhin stehen Funktionen zur Behandlung von *text windows*, *canvas windows* (das sind Grafik-Fenster mit einem vordefinierten Menu zur Parameterwahl und zur PostScript-Ausgabe) und *frame windows* (Fenster mit mehreren möglichen Subfenstern) zur Verfügung.

Im Vergleich zur *MiniGraphik* (s. Kap. 2.1.2 Seite 11) sind die Xgraphic-Funktionen also mehr auf Interaktion mit dem Benutzer ausgerichtet.

Die zentral installierte Version von Xgraphic benutzt als Toolkit die *Xt* und das *Athena Widget Set*, was von der Benutzerführung und Handhabung her sicherlich keine sehr komfortable Lösung darstellt, aber den Vorteil hoher Portabilität hat.

Die gegenwärtig installierte Version ist in konventionellem C implementiert, eine ANSI-Version (mit Strukturverbesserungen) ist inzwischen vorhanden und kann bei entsprechender Nachfrage bereitgestellt werden.

2.1.4 GKS

GKS, das *G*rafische *K*ern*s*ystem, ist eine genormte[3] Schnittstelle zur Erzeugung von 2-D-Computer-Grafik. In der Historie der Computergrafik machte es erstmals geräte- und maschinen*un*abhängige Implementierungen von grafischen Anwendungsprogrammen möglich.

GKS liegt in Form eines Unterprogrammpaketes vor und bietet somit eine prozedurale Schnittstelle für Grafik nutzende Anwendungsprogramme. Es erlaubt die Ausgabe zweidimensionaler Vektor- und Rasterbilder und unterstützt

[3]Internationale Norm: ISO 7942, deutsche Norm: DIN 66252

Bedienereingabe sowie Interaktion durch Funktionen für grafische Eingabe und Bildstrukturierung. Weiterhin ermöglicht es Speicherung und dynamische Veränderung von Bildern.

Heute existieren genormte Schnittstellen für verschiedene Sprachen. Am ZIB sind FORTRAN-77- und C-Versionen verfügbar.

Treiber für grafische Ausgabegeräte sind Bestandteil der GKS-Bibliothek. Sie bieten sowohl die unmittelbare Ausgabe (und auch Eingabe) der durch ein Anwendungsprogramm erzeugten Bilder, als auch deren Ausgabe in eine Datei (sogenanntes *„Metafile“*) zur Langzeitspeicherung. An diesem Punkt kommt es häufig zu Mißverständnissen: In der GKS-Norm ist nur die prozedurale Schnittstelle zur Aus- und Eingabe von Bilddateien beschrieben, nicht aber das eigentliche Format der Bilddatei. Zwar wird in einem Anhang zur Norm ein GKS-Metafile (GKSM) definiert, doch ist dies nicht Teil der Norm selbst, so daß die Metafile-Formate der unterschiedlichen GKS-Implementierungen fast immer differieren. Genau genommen enthält ein GKS-Metafile kein statisches Bild, sondern eine Art Sitzungsprotokoll, in dem alle grafischen Ausgaben, die an das Ausgabegerät (Bildschirm) gehen, festgehalten sind.

Ein genormtes Format für eine echte 2-D-Bilddatei ist das CGM (**C**omputer **G**raphics **M**etafile). Grafische Geräte wie z.B. Plotter oder Sichtgeräte werden in der GKS-Sprechweise genauso wie das Metafile als *Workstation* – im folgenden GKS-Workstation genannt – bezeichnet. Für eine vorhandene Konfiguration von Ausgabegeräten ist eine spezifische GKS-Implementierung vorzunehmen durch Einbinden entsprechender Gerätetreiber aus der GKS-Bibliothek.

Aufgrund der Möglichkeit, mehrere GKS-Workstations gleichzeitig zu aktivieren, kann man zusätzlich zum Bildschirm auch ein Metafile „mitlaufen“ lassen, auf welchem alle Ausgaben protokolliert werden.

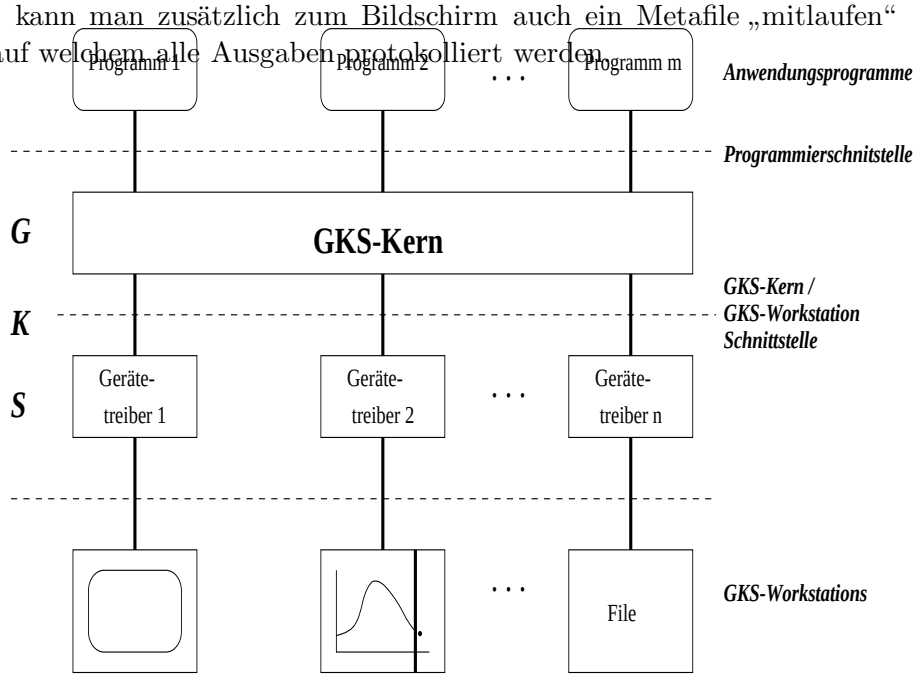

Abbildung 2.2: GKS-Systemarchitektur

Der „normale“, programmierende Benutzer braucht nur Kenntnisse über die Anwenderschnittstelle in der gewählten Sprache, und er muß die vorhandenen GKS-Workstations seiner GKS-Implementierung sowie deren Funktionalität kennen.

GKS stellt Funktionen sowohl für die Ausgabe als auch grafische Eingabe zur Verfügung. Die grafischen Primitive umfassen Linien (polylines), Markierungssymbole (polymarker), Flächen (fill area), Text und Zellmatrix (cell array). Für jedes dieser Primitive existiert ein Satz von Funktionen, der es erlaubt, zugehörige Attribute (Farbe, Strichstärke, ...) zu setzen oder abzufragen.

Für die grafische Eingabe stehen verschiedene Klassen von logischen Eingabegeräten zur Verfügung. Diese Eingabegeräte werden über GKS-Funktionsaufrufe angesprochen. Damit kann man sowohl Texteingabe (über die Tastatur), als auch Koordinateneingabe (z.B. über Mausklick in einem mit GKS erzeugten Bild) bewerkstelligen.

Im GKS-Konzept ist eine vollständige Kontrolle des Ausgabegerätes durch GKS vorgesehen, was aber in einer Umgebung mit Bildschirmfenstern (z.B. im X-Window-System) nicht der Fall ist. Dadurch kann es zu gewissen Inkonsistenzen kommem. Beispielsweise kann der Benutzer ein Fenster, in welchem mit GKS ein Bild gezeichnet wurde, in der Größe verändern, ohne daß GKS darüber informiert wird, d. h. weiter von der ursprünglichen Größe ausgeht.

Im ZIB sind zur Zeit zwei GKS-Implementierungen vorhanden:

- **FUGKS** oder **NewGKS**. Hierbei handelt es sich um eine (ursprünglich an der FU entwickelte) Implementierung, die nur über eine FORTRAN-77-Schnittstelle verfügt. In dieser Implementierung sind die meisten Treiber enthalten. Unter anderem kann direkt PostScript- oder HPGL-Output erzeugt werden. Die Ausgabe auf SUN-Bildschirme erfolgt z.Z. noch über SunView [4].

- **XGKS**. Eine Public-Domain-Implementierung auf Basis der Xlib, die sowohl eine FORTRAN-77- als auch eine C-Schnittstelle beinhaltet. Sie bietet den Vorteil der Ausgabe in ein X-Fenster, kann daneben aber nur noch in ein CGM oder GKSM schreiben. Diese Formate müssen zur Ausgabe auf Papier nach PostScript gewandelt werden (Siehe Kap. 3.1.2, Seite 56). Hinsichtlich der Einhaltung des Standards und der beschriebenen Funktionalität einzelner Routinen ist diese Implementierung weniger zuverlässig als FUGKS.

Derzeit wird die GKS-Norm in internationalen Normungsgremien einer Revision unterzogen. Das zukünftige GKS soll über mehr grafische Primitive verfügen und insgesamt etwas moderner sein. Ob ein revidiertes GKS am ZIB zur Verfügung stehen wird, ist nicht sicher.

[4] Ein X-Treiber wird derzeit am ZIB erstellt.

2.1.5 PHIGS, PHIGS PLUS und PEX

PHIGS

PHIGS (*P*rogrammer's *H*ierarchical *I*nteractive *G*raphics *S*ystem) ist eine prozedurale Schnittstelle (Unterprogrammbibliothek) zur Erzeugung von *drei*dimensionalen Grafiken [35]. Es stellt, ebenso wie GKS, eine internationale Norm[5] dar. Nach GKS entstanden, bietet PHIGS neben 3-D-Funktionalität viele weitere Features und ist konzeptionell etwas moderner.

Im Gegensatz zu *Xlib* gibt es in *PHIGS* keine Funktionen, mit denen direkt ein Daten-Primitiv dargestellt werden kann. Statt dessen bietet *PHIGS* Funktionen zum Edieren sogenannter *Strukturen*. Eine Struktur ist eine Folge von *Strukturelementen*, die Grafik-Primitive (z.B. Linien oder Gitter) Attribute (z.B. Farbe oder Rotation) oder andere Strukturen sein können. Letzteres bedingt das 'hierarchical' im Namen.

Wie wird nun aus einer Struktur eine Grafik? Dazu bietet *PHIGS* sogenannte *Workstations*, die jeweils zu einem Fenster oder anderen Ausgabemedien gehören. Gibt man nun einer Workstation eine Struktur, so wird diese traversiert und jedes Grafik-Primitiv mit den jeweils gültigen Attributen ausgegeben. Eine *PHIGS-Struktur* entspricht somit einem Grafik-Programm, das von einer *PHIGS-Workstation* interpretiert wird. Stukturen als Strukturelemente entsprechen dabei dem Anfang eines Unterprogramms.

PHIGS-Programmierschnittstellen wurden für verschiedene Sprachen definiert; die FORTRAN-77- und C-Programmierschnittstelle sind genormt.

PHIGS PLUS

Die Qualität mit PHIGS berechneter Bilder wird heute oft als nicht mehr ausreichend betrachtet. Um Anschluß an heutige Qualitätsstandards zu gewinnen, wurden zu PHIGS Erweiterungen definiert. Die erste Erweiterung, PHIGS PLUS, ist inzwischen Bestandteil des PHIGS-Standards (ISO), die zugehörige C-Programmierschnittstelle ist noch Gegenstand von Normungsaktivitäten [35]. PHIGS PLUS enthält im wesentlichen folgende Erweiterungen gegenüber PHIGS: grafische Primitive für Oberflächen und Kurven, Beleuchtung, Schattierung, Depth Cueing[6] und Anti-Aliasing.

Je nach Implementierung unterstützen auch PHIGS und PHIGS PLUS verschiedene Ausgabegeräte. Daher sind PHIGS- bzw. PHIGS-PLUS-Programme trotz genormter Programmierschnittstelle nicht ohne weiteres portabel. Die Norm für die C-Schnittstelle (nur PHIGS, nicht PHIGS PLUS) wurde vor einiger Zeit verabschiedet. Zur Zeit sind noch nicht alle PHIGS-Implementierungen hinsichtlich ihrer C-Programmierschnittstelle auf dem neuesten Stand.

PEX

Um PHIGS netzweit nutzen zu können, beispielsweise auf einem *entfernten* Rechner ein Programm zu starten, das mit PHIGS *lokal* 3-D-Bilder darstellt,

[5] ISO 9592

[6] s. Anhang B

wurde unter Leitung des MIT das PEX (*P*HIGS *E*xtensions on *X*) entwickelt. Dies war erforderlich, da PHIGS-Implementierungen 3-D-Primitive unter Berücksichtigung ihrer Attribute zunächst nach 2-D abbilden müßten, um das X-Protokoll zu nutzen. Mit PEX dagegen werden 3-D-Primitive und ihre Attribute über das Netz transportiert und am Ausgabeort durch einen erweiterten X-Server, den *PEX-Server*, interpretiert.

Der Strukturspeicher kann bei PEX sowohl auf der Client-, als auch auf der Serverseite liegen. Ein verteilter Strukturspeicher ist ebenfalls möglich. Wie X definiert PEX keine Programmierschnittstelle, sondern ein Netzwerkprotokoll, dessen Funktionalität ähnlich zu der von PHIGS oder PHIGS PLUS ist. Daher wird PEX üblicherweise über eine PHIGS-Programmierschnittstelle angesprochen, aber auch andere Programmierschnittstellen sind vorhanden oder in Entwicklung. Vom X-Consortium wird derzeit, in Analogie zur Xlib, die PEXlib entwickelt [48], eine Programmierschnittstelle, die „näher“ am PEX-Protokoll liegt als PHIGS. Diese Neuentwicklung erlaubt eine bessere Integration in die X-Welt, als es mit PHIGS möglich ist.[7] PEX entfernt sich somit von PHIGS, was seine Durchsetzungsfähigkeit am Markt aber eher verbessern wird. PHIGS ist zwar im Namen PEX verankert, doch könnte dies in einiger Zeit nur noch von historischer Bedeutung sein.

Die Entwicklung von PEX ist nicht abgeschlossen. Ebenso wie im X-Protokoll sind im PEX-Protokoll Erweiterungsmöglichkeiten vorgesehen, die in Zukunft sicher genutzt werden. So soll PEX etwa um zusätzliche Grafik-Primitive erweitert werden.

Am ZIB sind zur Zeit folgende PHIGS-, PHIGS-PLUS- und PEX-Implementierungen verfügbar:

- **SunPHIGS 2.0**. Die von Sun entwickelte PHIGS- und PHIGS-PLUS-Implementierung [22, 60, 59, 61, 58] steht auf allen Sun-Workstations des Hauses zur Verfügung. SunPHIGS enthält Treiber für X- und CGM-Output[8]. Zur Bildschirmdarstellung benutzt SunPHIGS die proprietäre Grafikbibliothek XGL. Es besteht die Möglichkeit, mit PHIGS-Funktionen in schon geöffnete X-Fenster zu zeichnen oder mit PHIGS selbst ein Fenster zu öffnen. Das Zeichnen in ein X-Fenster ist schneller. Man kann jedoch für die Eingabe ausschließlich das X-Input-Konzept nutzen und nicht - wie bei PHIGS-eigenen Fenstern - die grafischen Eingaben durch PHIGS-Funktionen realisieren.

  Papierausgaben können nur durch Ausgabe in ein CGM und nachträgliches Konvertieren nach PostScript erstellt werden. Das von der derzeitigen SunPHIGS-Version erzeugte CGM ist fehlerhaft, so daß die Qualität des daraus erzeugten Ausdrucks mangelhaft ist. Bevor dieser Fehler nicht beseitigt ist, kann SunPHIGS nicht empfohlen werden.

- **FIGARO+ 3.0**. Diese von der Firma Liant implementierte PHIGS-

[7] PHIGS wurde wie GKS zu Zeiten entwickelt, in denen X noch nicht populär war und die Steuerung des Bildschirms und der Eingabegeräte als Funktionen von Grafikbibliotheken implementiert wurde.

[8] CGM = Computer Graphics Metafile s. Kap. 3.1.2.

und PHIGS-PLUS-Version einschließlich firmenspezifischer Erweiterungen [66, 17, 18, 15] steht auf Silicon-Graphics-Workstations zur Verfügung. Die PHIGS-PLUS-Funktionen von FIGARO+ unterscheiden sich derzeit noch von den entsprechenden Funktionen in SunPHIGS. So sind sowohl die Funktionsnamen als auch die Anzahl und Struktur der Funktionsparameter in vielen Fällen unterschiedlich. Dadurch können mit SunPHIGS erstellte Programme, welche PHIGS-PLUS-Funktionen nutzen, nicht ohne weiteres mit FIGARO+ laufen. Die erwünschte Normkonformität der verschiedenen PHIGS-PLUS-Implementierungen wird sich jedoch mit der Zeit einstellen. Auch die Steuerung der Ausgabe ist unterschiedlich gelöst: SunPHIGS verwendet dazu spezielle Funktionen, während FIGARO+ extensiv von Umgebungsvariablen Gebrauch macht.

Im ZIB läuft FIGARO+ nur auf den Silicon-Graphics-Workstations grafs11 und graf13.

FIGARO+ nutzt zur Darstellung auf dem Bildschirm die IRIS Graphics Library (GL) (siehe Kap. 2.1.6, Seite 17). Um auf andere Geräte ausgeben zu können, muß man die zugehörige, auch am ZIB vorhandene Peripheral Support Option (PSO) benutzen [16]. Dann stehen u.a. CGM, HPGL und Colour PostScript als Ausgabeformate zur Verfügung:

- **PEX**. Die am ZIB installierte PEX-Implementierung ist die von Sun erstellte und nun vom X-Consortium (MIT) erhältliche PEX-SI (*PEX Sample Implementation*). Die Programmierschnittstelle ist im wesentlichen die gleiche wie bei Sun-PHIGS. Allerdings ist nur Bildschirmausgabe möglich. Weiterhin ist keine FORTRAN-Schnittstelle vorhanden.

  Um PEX-SI nutzen zu können, muß der erweiterte X-Server auf der ausgebenden Maschine laufen. Mit

  ```
  xdpyinfo | grep X3D-PEX
  ```

  kann man feststellen, ob der entsprechend erweiterte Server auf dem jeweiligen Rechner läuft. Ist das nicht der Fall, so muß vor dem Aufruf von OpenWindows

  ```
  setenv SERVER /usr/bin/X11/X
  ```

  gesetzt werden. PEX-SI hat generischen Charakter und ist nicht auf irgendwelche Hardware zugeschnitten, daher aber langsamer als hardwareabhängige, kommerzielle Produkte. Dies ist beabsichtigt: kommerzielle Implementierungen, insbesondere der Rechnerhersteller, sollten nicht ihres Marktes beraubt werden. Weitere derzeit verfügbare PEX-Programmierschnittstellen sind Xpex (Sony), pexim (Stardent), PEXt (Evans & Sutherland) und dec-pexlib.

2.1.6 GL, DGL und OpenGL

Die *G*raphics *L*ibrary (*GL* oder *IRIS GL*) ist eine von Silicon Graphics entwickelte Bibliothek. Sie stellt einen guten Kompromiß zwischen theoretisch

erwünschter und effizient in Hardware realisierbarer Funktionalität dar. Die GL ist eine Basisbibliothek mit vielen, meist einfachen Funktionen, die jedoch, richtig eingesetzt, die Erzeugung hochwertiger Grafiken erlauben. Programmierschnittstellen sind derzeit für C- und Fortran-77 definiert. Neben den üblichen Basisgrafikelementen Punkte, Linien, Flächen, einfache Texte, grafische Eingabe, und deren Attributen, sind auch Funktionen wie

- NURBS[9]-Linien
- NURBS-Flächen
- Depth Cueing
- Anti-Aliasing
- Beleuchtung
- Schattierung
- Transparenz
- Texturen[10]
- Nebeleffekte

verfügbar. Z-Buffer, Double Buffering und True Colour[11] sind ebenfalls selbstverständliche Features. Je nach Ausbaustufe der Grafik-Hardware werden alle diese Möglichkeiten direkt in der Hardware oder durch Software-Emulation verfügbar gemacht. Die GL bietet auch ein Konzept zur Erstellung von grafischen Objekten aus mehreren Elementen, die dann als Ganzes behandelt, z.B. transformiert, werden können. Allerdings sind die damit erreichbaren Möglichkeiten nicht mit dem Strukturmodell von PHIGS vergleichbar. In der GL gibt es derzeit neben der Bildschirmsteuerung keine weiteren Gerätetreiber, so daß Ausgaben auf andere Medien nur über Screendumps möglich sind.

Mit der GL kann man auch über das Netz Grafiken auf anderen Bildschirmen erzeugen, sofern die beiden beteiligten Maschinen SGI-Workstations sind oder die DGL (*D*istributed *GL*) unterstützen. Dabei werden auf einer Maschine abgesetzte GL-Funktionsaufrufe mit Parametern gemäß einem SGI-eigenen Protokoll an eine zweite Maschine gesendet, wo sie durch Umsetzung in lokale GL-Funktionsaufrufe die Bildschirmausgabe bewirken. Zwischen den am ZIB vorhandenen SGI-Workstations ist die DGL nutzbar.

Die Stellung von Silicon Graphics als langjähriger Marktführer auf dem Gebiet der Grafik-Workstations und die ständige Weiterentwicklung der GL sorgte für eine weite Verbreitung der GL, so daß diese inzwischen als Industriestandard bezeichnet werden kann. Moderne höhere Grafikbibliotheken, Visualisierungsumgebungen und Animations-Software bauen zu einem nennenswerten Teil auf der GL auf. Andere Hard- und Software-Hersteller haben die GL lizensiert und bieten sie auf ihren Maschinen an (z.B. IBM). Um die Bibliothek

[9] **N**on-**u**niform **r**ational **B**-**S**plines.
[10] s. Texturen, Depth Cueing, Anti-Aliasing im Glossar, Anhang B.
[11] s. True Colour im Glossar, Anhang B.

für einen weiteren Kreis von Herstellern interessant zu machen, wird derzeit unter Federführung von Silicon Graphics die *OpenGL* entwickelt. Neben Silicon Graphics sind weitere Soft- und Hardware-Firmen[12] beteiligt. Diese Bibliothek ist aus der GL abgeleitet, doch wurden alle Window-Management- und Eingabe-Routinen herausgenommen, um Anwendungsprogramme von Hardware, Betriebs- und Windowsystem unabhängiger und somit portabler zu machen. Daneben wurden auch die Funktionsnamen vereinheitlicht. Des weiteren soll es Möglichkeiten geben, nicht nur direkt auf dem Bildspeicher zu *rendern*[13], sondern auch in den normalen Hauptspeicher (allerdings unter Verzicht auf Unterstützung durch Grafik-Hardware). Damit wird dann Berechnung von Bildern höherer Auflösung und eine Abspeicherung der Bilder in Rasterdateien möglich. Eine Erweiterung zur OpenGL ist die GLX[14] (Open *GL* e*x*tension), womit die OpenGL innerhalb von X-Window genutzt werden kann.

Sobald sich andeutet, daß die – derzeit nur in Konkurrenz zu PEX stehende – OpenGL-Bibliothek sich auf dem UNIX-Markt durchsetzt, wird sie auch am ZIB verfügbar gemacht werden.

Literatur zu OpenGL: s. [47, 41].

2.1.7 Inventor

IRIS Inventor ist eine objektorientierte 3-D-Grafikbibliothek. Sie bietet eine Vielfalt von Funktionen, mit denen die Entwicklung komplexer, insbesondere auch interaktiver 3-D-Grafiksoftware unterstützt wird. Inventor baut auf der GL (vgl. Kap. 2.1.6) auf und erweitert sie zu einem sehr mächtigen Werkzeug, mit dem komplexe dreidimensionale „Szenarien“ oder „Landschaften“ aus geometrischen Modellen in einfacher Weise erzeugt, dargestellt und verändert werden können. Der Schwerpunkt von Inventor liegt nicht darin, neue Grafikoperationen und Darstellungsformen zur Verfügung zu stellen. Diese bleiben auf dem Stand der GL. Was Inventor bietet, ist vielmehr ein Baukastensystem zur Einbettung anspruchsvoller 3-D-Grafik in eigene Applikationen.

Das Konzept der Bibliothek ist vollständig objektorientiert. Sie ist in C++ geschrieben. Es bietet sich deshalb an, die Inventor-Funktionen auch aus C++-Programmen heraus zu nutzen. Eine C-Anbindung ist zwar ebenfalls möglich, allerdings ist auch für C-Programmierer ein Hineindenken in die objektorientierten Strukturen unumgänglich.

Zur internen Repräsentation einer Szenerie ist in Inventor eine Baumstruk-

[12] Das über Definition, Einheitlichkeit und weitere Entwicklung der OpenGL wachende „Architectural Review Board“ besteht derzeit aus SGI, DEC, IBM, Intel und Microsoft. Die OpenGL soll nicht nur im UNIX-Markt plaziert werden, sondern ist auch als Grafik-Standardbibliothek für Windows NT, also für den PC-Markt gedacht.

[13] s. Rendering im Glossar, Anhang B.

[14] Die GLX stellt zweierlei dar: erstens, die Definition eines Netzprotokolls zur Kommunikation zwischen einem Client und Server in einer X11-Umgebung (wie bei PEX: eine Erweiterung des X11-Protokolls und implementiert mit den gewöhnlichen in X11 vorgesehenen Erweiterungsmechanismen); zweitens einen Satz von Funktionen zur Integration der OpenGL in eine X11-Umgebung (Kommunikation mit dem X-Server, Nutzung der X-Fonts, Synchronisation, usw.).

tur vorgesehen, der sogenannte *Scene Graph* . Entsprechend dem objekt-orientierten Konzept der Bibliothek ist dieser aufgebaut aus „Objekten“ folgender Art:

- *shape*-Objekte, die geometrische Modelle repräsentieren, zum Beispiel Linien, Kugeln, polygonale Oberflächen, NURBS[15] oder Text
- *property*-Objekte, mit denen die Eigenschaften der geometrischen Modelle definiert werden können, zum Beispiel Farbe, Transparenz, Texturen oder geometrische Transformationen
- *group*-Objekte, die zur Anordnung und Zusammenfassung von Objekten zu Gruppen dienen, denen dann bestimmte Eigenschaften als Ganzes zugewiesen werden können
- *camera*- und *light*-Objekte, mit denen Beleuchtung und Perspektive definiert werden.

Die einzelnen Knoten in einem *Scene Graph* bilden einen DAG (directed acyclic graph). Bei der Darstellung der Szenerie wird der Graph in einer definierten Weise traversiert. Dabei wirken *property*-Objekte beispielsweise nur auf nachfolgende Knoten. Die Funktionalität des Graphen und somit das endgültige Bild folgt also aus der Art und Anordnung der einzelnen Knoten innerhalb der Baumstruktur.

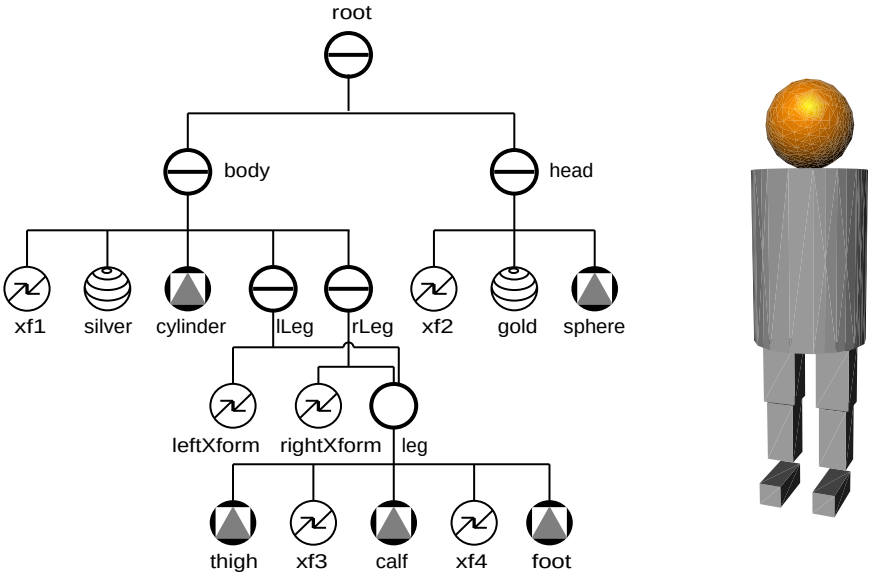

Abbildung 2.3: Beispiel für einen Inventor *Scene Graph*, durch den die nebenstehende einfache Figur erzeugt wird.

Neben den einfachen Objekten existieren auch sogenannte *Manipulatoren*, mit deren Hilfe sich in einem Scene Graph interaktiv Veränderungen vornehmen

[15] **N**on-**u**niform **r**ational **B**-**S**plines.

lassen, z.B. die *handle box*: Man wählt per Mausklick ein bestimmtes geometrisches Objekt innerhalb der Szenerie aus, dann wird eine *handle box* aufgerufen und automatisch ein Rahmen um das entsprechende Objekt gezeichnet; der Benutzer kann den Rahmen mit der Maus „anfassen“ und das Objekt auf diese Weise zum Beispiel verschieben oder vergrößern.

Weiterhin stehen in Inventor Routinen zur Verfügung, die beispielsweise eine interaktive Farbauswahl in einem eigenen Fenster ermöglichen (*Color Editor*), oder mit denen ein kompletter Scene Graph dargestellt, gedreht und verschoben werden kann (*Examiner Viewer*). Inventor-Applikationen lassen sich sowohl mit einer X- als auch GL-basierten grafischen Benutzungsoberfläche versehen.

Die Fülle grafischer Operationen, die von der Bibliothek bereitgestellt werden, kann im einzelnen hier nicht erörtert werden. Oft handelt es sich um eine Zusammenfassung einzelner GL-Routinen zu mächtigeren Aufrufen. Ein *Scene Graph* kann sowohl als Textfile als auch binärkodiert gespeichert werden. Weiterhin läßt sich Inventor-Code in einfacher Weise innerhalb der Visualisierungsumgebung IRIS Explorer (vgl. Kap. 2.3.2 Seite 29) nutzen, da sich hinter dem Geometrie-Datentyp, der in Explorer für 3-D Objekte verwendet wird, ein Inventor *Scene Graph* verbirgt.

Literatur zu Inventor: s. [57].

2.1.8 NAG Graphics Library

Die NAG Graphics Library ist eine Unterprogrammbibliothek mit FORTRAN-Schnittstelle zur Erzeugung zwei- und dreidimensionaler Präsentationsgrafiken. Es stehen Routinen für

- Achsen, Gitter, Rahmen, Beschriftung
- Punkte und gerade Linien
- grafische Darstellung der Lösung gewöhnlicher Differentialgleichungen
- allgemeine Funktionen mit einer oder zwei Variablen
- Höhenlinien
- Isometrische und perspektivische Oberflächen
- Vektorfelder
- statistische Grafiken

zur Verfügung.
Die NAG Graphics Library besteht aus zwei Software-Schichten. Die High-Level-Routinen gehören zur geräteunabhängigen Schicht. Über das NAG Graphical Interface erfolgt die Grafikausgabe entweder direkt über eingebaute Gerätetreiber (z.B. PostScript) oder durch den Aufruf von Funktionen einer Basisgrafikbibliothek (z.B. GKS). Am ZIB werden Graphical Interfaces für PostScript-Ausgabe und für GKS unterstützt.

Literatur zu NAG: [40].

2.2 Plotpakete

In wissenschaftlichen Anwendungsbereichen stellen Programme zur interaktiven Erstellung von Kurvenplots und anderen Diagrammen weiterhin zentrale grafische Werkzeuge dar. Die Darstellung von $R \to R$ oder $R^2 \to R$ Funktionen stellt die einfachste Art der „wissenschaftlichen Visualisierung“ dar. Trotzdem gibt es in der UNIX-Workstation-Welt bisher kein Pendant zu den in der DOS/Windows-Welt verbreiteten, leistungsfähigen und recht preisgünstigen kommerziellen Plotprogrammen. Als UNIX-Anwender ist man entweder auf deutlich überteuerte kommerzielle Systeme oder aber auf eigene Programme und Public-Domain-Programme angewiesen, wenn man nicht von Systemen für spezielle Anwendungsbereiche, wie etwa dem von NCAR für die Meteorologie und Klimaforschung, Gebrauch machen kann. Für Anwendungen, die aus eigenen Programmen gesteuert werden, kann man auch Funktionen aus höheren Grafik-Bibliotheken (z.B. NAG) nutzen.

Am ZIB gibt es mehrere Stand-Alone-Programme zur Erzeugung von Plots. Das im Hause vor mehreren Jahren nach Vorgaben aus der Numerik, insbesondere zum flexiblen Eingabeformat, entwickelte Programmpaket Grazil (2-D-Version) und Grazil3D (3-D-Version) [38] basiert auf GKS, Grazil3D auf PHIGS PLUS [16]. Grazil bietet den Vorteil, daß die Entwickler im Haus sind und somit Sonderwünsche, Verbesserungen sowie gezielte Weiterentwicklungen realisieren können. Die Bedienung von Grazil wurde in jüngster Zeit durch eine grafische Benutzerschnittstelle vereinfacht.

Besonders beliebt ist das weltweit verbreitete und für eine riesige Palette von Hardware-Plattformen verfügbare Public-Domain-Paket GNUPLOT. Es läßt sich so einfach bedienen, daß man als unerfahrener Benutzer schon nach wenigen Minuten zu ersten Bildern kommt. Die Praxis zeigt, daß Erweiterungswünsche durch eigenhändige Modifikation des Source-Codes oft recht schnell erfüllbar sind. Daneben ist am ZIB das X-basierte Public-Domain-Programm ACE/gr (Openlook-Version: xvgr, Motif-Version: xmgr) verfügbar. Ein schönes Feature dieses sonst recht einfachen Programmes ist die Möglichkeit, Kurven zu editieren, etwa durch Hinzu- oder Wegnahme von Punkten. Eine weiterer Weg für die Erstellung von Plots ist, in mathematische Anwendungsprogramme (etwa Matlab) eingebaute Plot-Möglichkeiten nutzen – auch wenn die darzustellenden Daten „außerhalb“, d.h. mit anderen Programmen, erzeugt wurden.

Empfehlungen: Die Möglichkeiten sowie Stärken und Schwächen der einzelnen Pakete sind zu vielfältig und die Unterschiede zu groß, als daß ein pauschales Urteil möglich wäre. Vorschlag für Mitarbeiter, die sich noch nicht an eines der genannten Plot-Pakete gewöhnt haben: Falls man im Anwendungsprogramm (wie etwa in Matlab) über Plot-Möglichkeiten verfügt, zunächst diese nutzen. Die nächste Wahl könnte GNUPLOT sein. Kann dieses die Wünsche nicht erfüllen, sollte man es mit Grazil versuchen. Falls auch dieses nicht ausreicht, in der Visualisierungsgruppe weitere Möglichkeiten erfragen!

Wegen häufiger Probleme in der Vergangenheit sei hier eigens nochmal dar-

[16] Eine begonnene 3-D-Implementierung auf Basis der GL wurde aus verschiedenen Gründen vorläufig gestoppt.

auf hingewiesen: wie bei den meisten UNIX-Programmen gilt auch bei den Plot-Programmen, daß die Environment-Variablen (der jeweiligen Dokumentation zu entnehmen) richtig gesetzt sein müssen.

Ein weiterer genereller Hinweis: beim Erzeugen von Plots für Publikationen von vornherein an die beim Setzen für den Druck in der Fachzeitschrift vorgenommene Verkleinerung denken; Plots der Größe A4 werden oft auf wenige cm Seitenlänge geschrumpft; man sollte daher die Schrift überproportional groß wählen und auch für ausreichende Strichstärke sorgen. Wird die Grafik auf Video aufgezeichnet, gilt dies in noch stärkerem Maße.

2.2.1 GRAZIL und GRAZIL3D

GRAZIL

GRAZIL, ein Anwendungsprogramm zur Darstellung von Kurvenverläufen in der Ebene, ist ein fertiges Hauptprogramm auf der Basis von GKS. Im ZIB steht es derzeit nur auf SUN-Workstations zur Verfügung.
Es erlaubt zwei verschiedene Arbeitsweisen:

- **Postprocessing**
  Bereits vorhandene Daten aus einer Anwendung werden von einer Datei eingelesen und dann auf dem gewünschten Ausgabegerät grafisch dargestellt.

- **Online-Darstellung**
  Nachdem GRAZIL gestartet wurde, kann in einem anderem Fenster ein Anwendungsprogramm aufgerufen werden, das Daten erzeugt und (mittels zusätzlich eingebundener Funktionen) an GRAZIL sendet. GRAZIL stellt diese Daten sofort nach dem Empfang dar und speichert sie intern für spätere Wiederverwendung.

Arbeiten mit GRAZIL
GRAZIL dient der interaktiven oder script-gesteuerten Erstellung von Plots. Die Steuerung von GRAZIL erfolgt über eine Kommandoschnittstelle; eine Menü-Oberfläche wurde erstellt und steht kurz vor der Freigabe. Die bereitgestellten Befehle bieten unter anderem folgende Einstellungsmöglichkeiten:

- Einteilung von Achsen, deren Positionierung und Beschriftung
- Aufteilung der Zeichenfläche in bis zu 4 Darstellungsbereiche (Layer)
- Auswahl von Darstellungsmodi (Balkendiagramm, Kurve, Kurve mit Fehlerbalken)
- Auswahl von Legendentypen und Position der Legende
- Setzen von frei positionierbaren Überschriften
- Unterlegen der Zeichenfläche mit einem Gitter.

Alle Einstellungen können in eine Datei gesichert und wieder eingelesen werden. Die Daten werden *nicht* gesichert.

Die Struktur der Eingabedaten
Man kann 1-, 2- und 3-dimensionale Datenfelder einlesen, diese untereinander in Beziehung setzen und durch weitere Angaben deren Interpretation festlegen.

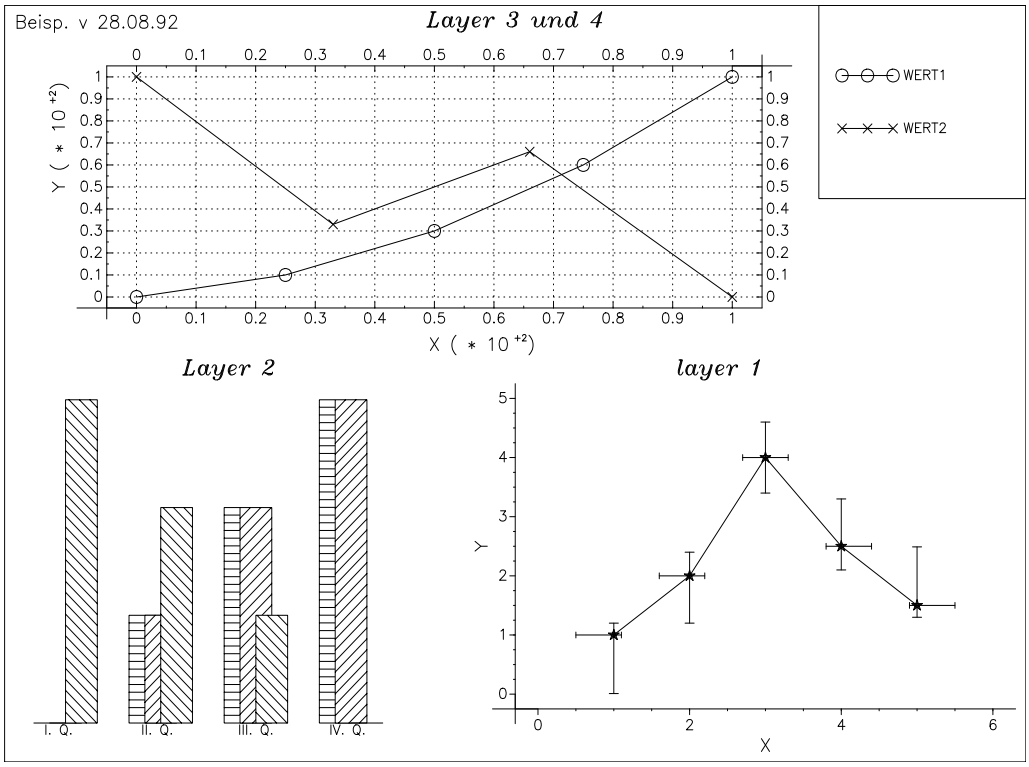

Abbildung 2.4: Beispiel eines mit GRAZIL erzeugten Bildes.

Ausblick
Ein neu entwickeltes Graphical-User-Interface für GRAZIL wird derzeit getestet; es wird den Umgang mit GRAZIL wesentlich vereinfachen. Weitere wünschenswerte Erweiterungen, wie etwa eine breite Auswahl guter Schrifttypen und das Setzen mathematischer Formeln in Texten lassen sich innerhalb des vorgegebenen Rahmens wegen GKS-interner Limitierungen leider nicht realisieren.

Empfehlungen
Für gute Plots anstelle der GKS-Textfonts die Adobe-Textfonts benutzen, d.h. durch GRAZIL-Kommandos den Textfont und die Textprecision auf beispielsweise 22 2 (Font: Helvetica)[17] einstellen. Eine weitere Möglichkeit, bessere Textfonts für Beschriftungen zu erhalten, ist die Verwendung des LaTeX-Makros

[17]Diese Fonts sind nur bei PostScript-Ausgabe verfügbar, am Bildschirm verschlechtert sich die Textausgabe.

`psfrag`. Dieses bietet sich besonders bei der Einbindung von Plots in LaTeX-Texte an (siehe Kap. 2.4.6)

GRAZIL3D

GRAZIL3D ist eine Erweiterung von GRAZIL zur Darstellung 3-dimensionaler Daten (Flchen, Kurven und Kurvenscharen). Es beruht auf dem Grafikstandard PHIGS PLUS.

Die PHIGS-PLUS-Implementierung von SUN unterstützt derzeit nur die Ausgabegeräte Bildschirm und CGM. Direkte PostScript-Ausgabe von GRAZIL3D ist daher nicht möglich; Die Kommandoschnittstelle von GRAZIL3D ist ähnlich zu der von GRAZIL; die Datenschnittstelle ist gleich (eventuell fehlende Komponenten von Datenpunkt-Vektoren werden auf 0.0 gesetzt). Der Kommandosatz wurde an die Erfordernisse von 3-D-Darstellungen angepaßt und umfaßt unter anderem:

- Einstellen des Betrachtungspunktes in der 3-D-Szene
- Auswahl der Interpretationsmodi (Curve, Triangle, Trianglestrip, Quadrilateral, Arrow), d.h. der Art und Weise, wie die Daten grafisch darzustellen sind
- Erzeugen von Konturlinien und -Flächen
- Einstellen der Schattierung von Flächenelementen
- Erzeugen von Farbtabellen
- Umkopieren von Datenkomponenten in eine andere Variable.

Mit Hilfe der Interpretationsart ARROW können $R^3 \rightarrow R^3$-Vektorfelder visualisiert werden. Durch Färben der Pfeile läßt sich gleichzeitig ein 3-D-Skalarfeld darstellen.

Ausblick
GRAZIL3D kann bei Bedarf auf SGI-Rechner portiert werden. Die dort vorhandene PHIGS-Implementation unterstützt auch direkte PostScript-Ausgabe, so daß auch GRAZIL3D-Bilder ohne Qualitätsverlust aufs Papier gebracht werden könnten. Je nachdem, wie sich der Bedarf an GRAZIL3D entwickelt, wird auch dieses mit einer grafischen Benutzerschnittstelle versehen.

2.2.2 GNUPLOT

GNUPLOT ist ein kommando-orientiertes Programm, das sowohl interaktiv als auch per Skript gesteuert werden kann. Es dient zur grafischen Darstellung von zwei- und dreistelligen Relationen $(x, y) \subset R^2$ bzw. $(x, y, z) \subset R^3$ durch isolierte Datenpunkte oder interpolierenden Kurven bzw. 2-D-Flächen. Die Komponenten der Tupel können durch Datenwerte oder symbolische Ausdrücke (plus Definitionsbereich) gegeben sein. Möglich sind Parameterdarstellungen $(x(t),\ y(t))$

bzw. $(x(u,v),\ y(u,v),\ z(u,v))$, oder, bei (rechtseindeutigen) Abbildungen, explizite Darstellungen der Form $y = f(x)$ bzw. $z = f(x,y)$ [18].

Die Datenwerte können interaktiv eingegeben, aus einer Datei gelesen oder über UNIX-Pipes aus einem beliebigen Programm eingespeist werden. Die Tupel (x,y) bzw. (x,y,z) müssen hierbei in einem zeilenorientierten Format vorliegen. Das Format innerhalb einer Zeile ist sehr flexibel, da beim Lesen scanf-Formatangaben verwendet werden können. Auch binärkodierte Daten können gelesen werden. Durch Auswahlfunktionen können beliebige Datenspalten den Komponenten der Tupel zugewiesen werden. Wahlweise können die Daten auch in Zylinder- oder Polarkoordinaten angegeben werden.

Funktionen werden in C-ähnlicher Notation angegeben, wobei man von den üblichen logischen und arithmetischen Operatoren (unären und binären) sowie den Funktionen der C-Mathematik-Bibliothek libm Gebrauch machen kann. In Erweiterung zur Programmiersprache C existiert ein Datentyp complex. Außerdem sind alle mathematischen Funktionen sowohl für ganzzahlige, reell-dezimale und komplexe Argumente definiert.

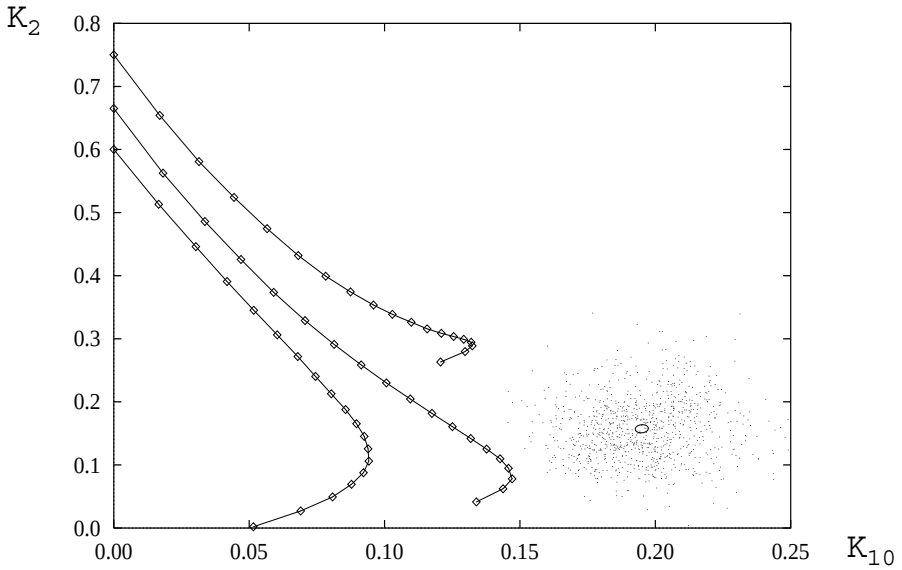

Abbildung 2.5: Beispiel eines mit *gnuplot* erzeugten Bildes

Bei der grafischen Ausgabe können Punkte, Linien und Impulslinien verwendet werden. Daneben gibt es für (x,y)-Daten weitere Darstellungsmöglichkeiten, wie etwa Fehlerbalken in y-Richtung und Kästchen zur Erstellung von Histogrammen. Die Darstellung kann in Polarkoordinaten erfolgen. Auch logarithmische Achsen sind möglich. Flächen werden durch Gitternetze dargestellt, wahlweise mit *hidden line removal*, Schattierung gibt es jedoch nicht. Funktions-

[18] Die Darstellungen $y = f(x)$ und $z = f(x,y)$ sind natürlich Spezialfälle der Parameterdarstellungen $(x(t),\ y(t))$ bzw. $(x(u,v),\ y(u,v),\ z(u,v))$ mit $x(t) = id_t$ bzw. $x(u,v) = id_u$ und $y(u,v) = id_v$.

werte an nicht-uniform verteilten Stützstellen können durch Interpolation mit wählbaren Abstandsnormen (L_{2^n}) an den Punkten eines regulären kartesischen Gitters berechnet werden. Für 3-D-Daten lassen sich Höhenlinien mit wählbaren Interpolations- bzw. Approximationseigenschaften (lineare Interpolation oder Interpolation durch kubische Splines, Approximation durch B-Splines) berechnen. Die Legende wird automatisch erzeugt. Zusätzlich können weitere Texte beliebig positioniert werden. Achsen werden von GNUPLOT automatisch erzeugt. Die Voreinstellungen für Achsentexte, Markierungen und Gitterlinien können vom Benutzer verändert werden. Alle Einstellungen können gesichert und so später weiter verwendet werden.

GNUPLOT unterstützt sehr viele Grafikformate. Die wichtigsten sind:

- **x11**, Ausgabe in ein X11-Fenster
- **iris4d**, GL-basierte Ausgabe auf einer Silicon Graphics Workstation
- **postscript** und **encapsulated postscript**
- **mif**, Ausgabe in eine Datei, die mit `Framemaker` weiter bearbeitet werden kann
- **pbm, pgm, ppm**, Ausgabe in eine Rasterbilddatei
- **latex**, **eepic**, Ausgabe für die LaTeX-Bildumgebung
- **fig**, Ausgabe in eine Datei, die mit `xfig` weiterbearbeitet werden kann.

Die mit GNUPLOT erzeugten Plots sind recht einfach gestaltet, reichen aber in vielen Fällen aus. Ein großer Vorteil ist, daß die Bilder mit dem Grafikeditor `xfig` weiterbearbeitet werden können.

Literatur zu GNUPLOT: [72].

2.3 Visualisierungsumgebungen

Die vielfältigen Anforderungen der visuellen Datenanalyse im wissenschaftlichen Bereich machten eine neue Kategorie von Software-Systemen erforderlich, sogenannte *Visualisierungsumgebungen*. Die Systeme müssen Werkzeuge zur klassischen Datenanalyse und -rekonstruktion, zur Bildsynthese, -analyse, -verarbeitung und -aufzeichnung enthalten, sowie über Archivierungs- und Datenbankfunktionen für Bilder, Filme und die Daten selbst verfügen. Wegen der Unmöglichkeit, den stark differierenden Anforderungen der vielen Anwendungsbereiche gerecht zu werden, müssen die Systeme auch erweiterbar sein. Die Konfiguration solcher Systeme muß selbst visuell unterstützt werden. Zur Einbettung in existierende Produktionsumgebungen ist eine direkte Kopplung an Anwendungsprogramme und eine Verteilung im Rechnernetz erforderlich.

Erste Vertreter dieser Software-Klasse, die zumindest einen Teil der Anforderungen erfüllen, sind in den letzten Jahren entstanden: es sind die Systeme *apE* [11], *AVS* [65], *Explorer* [32] und *Khoros* [52]. Die drei erstgenannten wurden (in der genannten Reihenfolge) zum Teil von denselben Entwicklern erstellt

und sind sich daher in vielerlei Hinsicht ähnlich. Khoros stellt eine unabhängige Entwicklung dar.

Am ZIB sind derzeit apE, Explorer und Khoros verfügbar [19]. Die Systeme sind jedoch so umfangreich, daß man sich unbedingt auf die Nutzung *eines* Produkts beschränken sollte. Die Visualisierungsgruppe am ZIB arbeitet hauptsächlich mit Explorer, dem von der Anlage her modernsten (aber noch etwas weniger ausgereiften) der drei Systeme. Das GL-basierte Produkt ist derzeit nur für SGI- und Cray-Rechner verfügbar[20], so daß die auf einer breiten Klasse von UNIX-Workstations lauffähige Urmutter apE weiterhin eine Daseinsberechtigung hat. Khoros weist gegenüber apE und Explorer besondere Stärken im Bereich der Bildverarbeitung auf.

Das neben Explorer in der Visualisierungsgruppe am ZIB meist verwendete System ist der *Advanced Visualizer* von Wavefront, ein aus der Computeranimation hervorgegangenes, komplexes Visualisierungssystem. Dessen Bedienung erfordert erhebliche Einarbeitungszeit und kontinuierliche Beschäftigung, so daß es für Gelegenheitsnutzer nicht geeignet ist.

Neben diesen „Generalisten“ gibt es auch kleinere, funktionell beschränktere und daher einfacher bedienbare Systeme. Hier sind das fertige Programm *SciAn* und das Framework *AGIL* einzuordnen. Des weiteren existiert eine Vielzahl von Systemen, die auf gewisse Anwendungen spezialisiert sind. Wir führen hier nur die am ZIB wirklich verwendeten Pakete *Grape* (Finite-Elemente-Methoden), *MPGS* (Stömungsmechanik und Strukturanalyse) und *Unichem* (Chemie) auf.

Es ist zu bemerken, daß dieses Kapitel die Vielfalt und Komplexität der Software-Landschaft in keiner Weise widerspiegelt. Es gibt eine Reihe weiterer, zum Teil nicht direkt vergleichbarer Systeme. Für den wissenschaftlichen Bereich beispielsweise *IDL*[21] (Interactive Data Language), das im Unterschied zu apE, AVS, Explorer und Khoros nicht „visual programming based“, sondern skript-basiert ist. Auf dem Markt der zum Wavefront Advanced Visualizer vergleichbaren Computeranimationssysteme herrscht Goldgräberstimmung; entsprechend vielfältig ist das Angebot. Angesichts der ständigen Neuerungen wird sich die Kontinuität im Softwarebereich, die wir eigentlich anstreben, nicht immer durchhalten lassen.

Empfehlungen: Existiert ein Spezialpaket, das auf Ihre Anwendung paßt, verwenden Sie dieses.[22] Ist das nicht der Fall, suchen Sie Zugang zu einer Silicon Graphics und freunden Sie sich mit Explorer an. Mit apE sollten nur noch Aufgaben in Angriff genommen werden, die sich ohne Programmierarbeit unmittelbar realisieren lassen. Die Entscheidung über die Weiterentwicklung von AGIL, das auf PHIGS PLUS basiert, wurde wegen der teilweise mangelhaften PHIGS-Implementierung von Sun und der Unvorhersehbarkeit der Entwicklung in der Frage PEX versus OpenGL zunächst verschoben. Es ist derzeit nur für einfache, schnell maßzuschneidernde Visualisierungsumgebungen auf Sun-Workstations

[19] Eine Beschaffung von AVS kann evtl. im Jahr 1994 realisiert werden.

[20] Sobald Explorer für andere Hardware-Plattformen, insbesondere Sun-Rechner verfügbar wird (was absehbar ist), erwägen wir auch den Einsatz auf anderen Rechnertypen.

[21] Derzeit nicht am ZIB verfügbar.

[22] Für den Finite-Element-Bereich wird erwogen, die in Grape vorhandene Funktionalität im Rahmen von Explorer zu realisieren.

zu empfehlen. SciAn, das durch seine einfache Bedienbarkeit hervorsticht, ist als abgeschlossenes, fertiges System nur dann zu empfehlen, wenn sicher ist, daß die darin vorgesehenen Darstellungsmöglichkeiten wirklich ausreichen.

2.3.1 apE

apE ist ein interaktives, modular aufgebautes Visualisierungspaket. Als reines Post-Processing System erhebt es den Anspruch, in einfacher Weise ohne Programmieraufwand und ohne Detailkenntnisse in der Computer-Grafik anspruchsvolle Darstellungen großer Datenmengen zu ermöglichen. Dazu werden verschiedene, jeweils unabhängige Module in geeigneter Weise in einer „Datenfluß-Pipeline“ miteinander verbunden. Das Produkt stammt von der Ohio State University und war bis zur Version 2.1 inklusive Quellen frei verfigbar.

Die Anfänge von *apE* gehen auf das Jahr 1987 zurück. Das System war das erste Allzweck-Visualisierungspaket, das auf dem Datenflußmodell aufbaute. Jüngere Systeme, die nach dem gleichen Prinzip arbeiten, sind *AVS* und *Explorer* (siehe Kap. 2.3.2 Seite 29). Von diesen Systemen steht Explorer ebenfalls am ZIB zur Verfügung. Im Vergleich zu Explorer ist apE naturgemäß weniger weit entwickelt. So ist zum Beispiel der Funktionsumfang der mitgelieferten Module bei apE geringer. Die Vorteile von apE sind folgende:

1. Der Quellcode ist verfügbar.
2. Das System läuft auch auf Sun-SPARC-Plattformen.
3. Alle Module sind von der Unix-Kommandozeile aus ausfiihrbar und lassen sich deshalb leicht in eigene Shell-Skripte einbinden.

Abbildung D.1 auf Seite 122 zeigt eine mit apE erzeugte Visualisierung aus dem Bereich der Strömungsmechanik.

2.3.2 Explorer

IRIS *Explorer* ist ein Softwarepaket, das auf die Visualisierung wissenschaftlicher Daten ausgelegt ist.

Das in Explorer verfolgte Konzept ermöglicht eine flexible Verarbeitung von Daten, die in beliebigen ASCII- und auch Binärformaten vorliegen, zu 2- oder (bevorzugt) 3-dimensionalen Grafikobjekten.

Solche Objekte können am Bildschirm betrachtet werden, wobei Einflußgrößen wie z.B. Standpunkt des Beobachters, Blickwinkel, Beleuchtungsmodelle, Vergrößerungsstufe interaktiv änderbar sind. Dadurch gewinnt man einen anschaulichen Eindruck von der aus den Eingabedaten berechneten Szenerie.

Die Verarbeitung der Eingabedaten geschieht in Explorer mittels sogenannter Module, die zu einer sogenannten *Explorer-Map* hintereinandergeschaltet werden.

Ein Modul ist ein - nicht eigenständig lauffähiges - Programm, das beliebige Eingabedaten auf Ausgabedaten abbildet (das Explorer-Datenformat umfaßt vier sehr allgemeine Explorer-Datentypen). Bei der Erstellung solcher Module

wird dem Benutzer vom Explorer-System sehr viel Arbeit abgenommen. Im optimalen Fall muß der Benutzer nur die gewünschte Abbildungsfunktion in C, C++ oder Fortran implementieren, den Rest der Modulerstellung übernimmt ein Hilfsprogramm, der Explorer *Module Builder*.

Verschiedene zueinander passende Module werden im Explorer *Map Editor* per Maus zu einer Explorer *Map*, d.h. einem Datenflußgraphen, zusammengefügt. An den Quellen des Graphen werden die Eingabedaten, die im Normalfall als Datei vorliegen, eingelesen und in ein Explorer-Datenformat konvertiert. Sie durchlaufen nun den Datenflußgraphen und werden dabei von jedem Modul in geeigneter Weise manipuliert, bis sie am Ende von einem Ausgabemodul dargestellt oder in ein Ausgabefile geschrieben werden. Bei den meisten Anwendungen wird dieses Ausgabemodul das sogenannte *Render-Modul* sein, das entstandene geometrische Objekte, wie oben angedeutet, interaktiv aus beliebigen Blickwinkeln darstellen kann. Ein Beispiel zum Map Editor wird in Abb. D.3 auf Seite 123 gezeigt; ein resultierendes Bild ist in Abb. 2.6 zu sehen.

Ein Modul präsentiert sich bei Programmablauf des Explorer-System innerhalb der Explorer-Map als grafisches Objekt oder als eigenes Fenster, in dem der Benutzer über automatisch verwaltete Bedienanzeigen Parameter einstellen und ändern kann.

Die Aufgabe, eigene Datenformate in für Explorer geeignete umzuwandeln, wird dem Benutzer durch das Hilfsprogramm *DataScribe* erleichtert. Das vorliegende Datenformat (ASCII oder binär) und das Explorer-Zielformat können hiermit interaktiv durch Manipulation graphischer Repräsentierungselemente beschrieben werden. Außerdem wird ebenfalls interaktiv eine Abbildungsfunktion definiert. Man erhält schließlich ein Explorer-Modul, daß die vorliegenden Daten einliest und automatisch konvertiert, so daß sie mit weiteren Modulen bearbeitet werden können.

Vordefinierte Module für allgemeine graphische Transformationen und Aktionen ermöglichen bereits komplexe Visualisierungen für so verschiedene Bereiche wie numerische Mathematik, Bilderzeugung, Molekularchemie, Atmosphärenphysik, Strömungsmechanik, Computer-Tomographie usw.

Durch das Modulkonzept erreicht Explorer eine sehr große Flexibilität, da immer mächtigere Module aufeinander aufbauen können, und je größer die eigene Modulbibliothek wird, desto vielfältiger werden die Anwendungsmöglichkeiten. Zugleich stellt dieses Konzept jedoch große Anforderungen an die Hardware, denn die zu verarbeitenden Datenmengen können bereits für überschaubare Anwendungen sehr große Ausmaße annehmen.

Die Abbildungen D.2 und D.4 auf den Seiten 122 *f* zeigen zwei Beispiele aus dem Bereich der Strömungsmechanik und der Finite-Element-Methoden. Eine Weiterverarbeitung von mit dem Explorer erstellten Objekten ist im Rahmen der Grafik-Bibliothek *Iris Inventor* möglich, da gleiche Formate zur Beschreibung von 3-D-Grafik-Szenerien (Geometry) benutzt werden (siehe Kap. 2.1.7, Seite 19).

IRIS Explorer ist ein Visualisierungswerkzeug, das von Anwendern mit unterschiedlicher Zielsetzung gewinnbringend einsetzbar ist. Die Zielgruppe umfaßt sowohl Wissenschaftler ohne Programmiererfahrung, die mit vorgefertig-

ten oder einfachen selbst erstellten Modulen ihre Daten visualisieren wie auch Programmierer, die – aufbauend auf den zur Verfügung gestellten Methoden – größere Visualisierungsprojekte realisieren wollen.

Das Konzept des Explorer erlaubt auch verteilte Anwendungen, d.h., es können Module auf anderen Workstations oder auch auf einer Cray laufen. Es sollten natürlich nur solche Module ausgelagert werden, die aufgrund ihres CPU-Bedarfs besser auf einem anderen Rechner laufen. Der Map Editor, von dem aus die Verteilung auf andere Rechner vorgenommen wird, ist dazu auf einer SGI-Workstation zu starten. Die grafische Ausgabe durch das Render-Modul erfolgt ebenfalls auf dieser Workstation. Der am ZIB vorhandene *Cray Explorer* erlaubt dem Anwender die Ausführung ausgewählter Explorer Module auf dem Vektorrecher *CRAY Y-MP2E/264*.

In der Version 1.0 des Cray Explorer werden nicht alle auf den SGI-Workstation verfügbaren Module unterstützt. Mit dem Module Builder erzeugte Module können ebenfalls in den Cray Explorer eingebaut werden. Mit DataScribe erzeugte Module werden vom Cray Explorer in der Version 1.0 nicht unterstützt. Literatur zu Explorer: s. [32, 31].

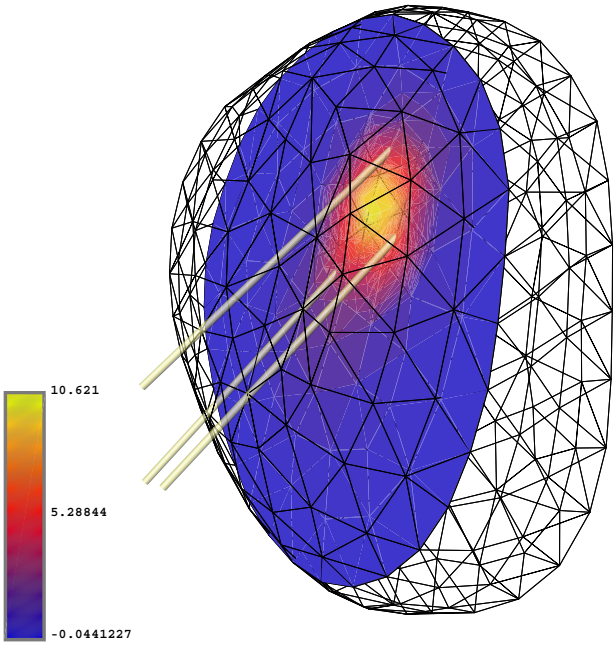

Abbildung 2.6: Mit Explorer berechnete Darstellung; erzeugt mit der Map aus Abb. D.3 (S.123) und dann in PostScript ausgegeben.

2.3.3 Grape

GRAPE (**GRA**phics **P**rogramming **E**nvironment) ist eine objektorientierte Programmierumgebung, die auf einfache Weise eine gute Auswahl an Visua-

lisierungsmöglichkeiten zur Verfügung stellt. Es ist besonders zugeschnitten auf den Finite-Elemente-Bereich, vom Prinzip her aber nicht darauf eingeschränkt. GRAPE erlaubt die Verknüpfung von Numerik und interaktiver grafischer Darstellung, ist flexibel und erweiterbar und besitzt eine geräteunabhängige Schnittstelle zur Grafikprogrammierung. Es wurde am Sonderforschungsbereich 256 am Institut für Angewandte Mathematik der Universität Bonn entwickelt.

Im wesentlichen wird GRAPE am ZIB zur Darstellung von zwei- und dreidimensionalen Finite-Element-Geometrien verwendet, wofür GRAPE spezielle Funktionen, wie etwa die Darstellung der verwendeten Triangulierungs- bzw. Tetraedisierungsfeinheit, zur Verfügung stellt. GRAPE benötigt dazu die Triangulierung bzw. die Tetraedisierung der Geometrie und die n-dimensionalen Lösungsvektoren. Zur Visualisierung stehen folgende Methoden zur Verfügung:

- Isolinien
- Isoflächen
- Darstellung der Triangulierung/Tetraedisierung
- Vektordarstellungen (z.B. Geschwindigkeit)
- Schnittebenen.

Es ist leicht möglich, eigene Methoden hinzuzufügen, so daß GRAPE an bestehende Anwendungen angepaßt werden kann. Eine eigene Methode kann z.B. das Einlesen benutzerspezifischer Daten sein (GRAPE schreibt kein bestimmtes Format für anwendungsbezogene Daten vor), oder ein weiteres Visualisierungsverfahren.

Zur grafischen Darstellung von Objekten bietet GRAPE Einstellmöglichkeiten für Licht und Materialeigenschaften, wie z.B. Lichtfarbe, Materialfarbe, Reflektion und Transparenz. Leider haben diese Angaben derzeit nur Einfluß auf die Bildschirmdarstellung, nicht aber auf die PostScript-Ausgabe. Darum sehen in PostScript ausgegebene GRAPE-Bilder (s.Anh.) anders aus als auf dem Bildschirm. Ebenfalls ist eine Ausgabe im Rayshade-Format (siehe Kap. 2.5.1, Seite 42) möglich. Leider sind auch hier wieder Unterschiede zur Bildschirmdarstellung vorhanden; sie entstehen, wie auch beim PostScript-Output, durch das Fehlen bestimmter Schattierungsverfahren. Die Darstellungs- und Kontrollfenster von GRAPE als Bildschirmabzug zeigt Abbildung D.5 auf Seite 124.

Literatur zu GRAPE: s. [71].

2.3.4 SciAn

SciAn (**Sci**entific **An**imation) ist ein interaktives Programm zur Visualisierung wissenschaftlicher Daten. Es wurde am Supercomputer Computations Research Institute der Universität Florida in Tallahassee entwickelt. Hervorstechende

Eigenschaft von SciAn ist die gute, teils selbsterklärende grafische Benutzerschnittstelle und das Online-Hilfskonzept. Dies ermöglicht es selbst dem Gelegenheitsnutzer, ohne Zuhilfenahme eines Handbuches schnell Daten zu visualisieren. SciAn liest Daten in verschiedenen Formaten ein. Das von den Entwicklern bevorzugte Format ist das Hierarchical Data Format (HDF) vom National Center of Supercomputing Applications in Urbana/Champaign (NCSA), welches am ZIB bisher nicht verwendet wird. Die anderen möglichen Datenformate sind das Gaussian90-Format und von diversen Anwendern implementierte Privatformate. Grundsätzlich stellen die Datenformate skalare oder vektorielle Daten über einem Gitter dar, wobei das Gitter auch gekrümmt oder unstrukturiert sein kann. Zeitabhängige Daten können ebenfalls verarbeitet werden.
Insgesamt stehen folgende Darstellungsmöglichkeiten zur Verfügung

- Isoflächen
- Konturen
- Vektorpfeile
- Traces
- Einfärbung
- Ball-and-Stick-Darstellungen von Molekülen
- einfache Geometrieobjekte.

Es können mehrere Datensätze gleichzeitig in einem Visualisierungsfenster oder es kann ein Datensatz in mehreren Visualisierungsfenstern dargestellt werden. Volumendaten können in 2-D-Schnitte unterteilt werden. Der Betrachterstandpunkt und Lichtquellen können ebenfalls gesetzt werden. Bei der Vielzahl von Einstellungsmöglichkeiten entsteht recht schnell eine Inflation an Fenstern auf dem Bildschirm, was die Übersichtlichkeit beeinträchtigt.

Alle erwähnten Einstellungen sind auch über Scripts steuerbar. Dadurch wird auch eine Animation der Visualisierung mit Aufzeichnung der Einzelbilder möglich. Bilder können als SGI-Rasterbild oder als PostScript-Datei abgespeichert werden.

Im Anhang D befinden sich zwei aus Demo-Daten gewonnene SciAn-Bilder (Abbildungen D.6 und D.7, Seite 124). Beide Bilder wurden als Screendumps erzeugt und dann weiterverarbeitet.

Literatur zu SciAn: s. [50].

2.3.5 AGIL

Mit *AGIL* steht dem Benutzer eine portable 3-D-Visualisierungsumgebung zur Verfügung. Weitgehende Portabilität ist durch die Nutzung von UNIX-Standards und die Wahl von PHIGS (siehe Kap. 2.1.5, Seite 15) als Basisbibliothek gewährleistet.

AGIL kann sowohl zum Postprocessing eingesetzt, als auch direkt an eine Anwendung gekoppelt werden, deren (Zwischen-)Ergebnisse dann parallel zur Berechnung visualisiert werden. Dazu bietet *AGIL* zwei Kommunikationsmechanismen: Standard-UNIX-Pipes (vgl. Abb. 2.7), die fast keine Veränderung der Applikation erfordern, und Shared-Memory-Kommunikation, wozu eine Funktionsbibliothek mit der Applikation gebunden werden muß. Der erste Mechanismus erlaubt auch eine Verteilung von Applikation und Visualisierung auf verschiedene Rechner.

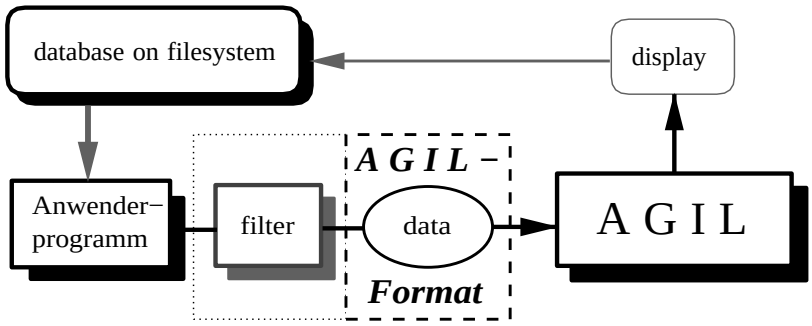

Abbildung 2.7: Interaktives Arbeiten mit *AGIL*

AGIL kennt sog. *Methoden*, mit denen eingelesene Daten manipuliert werden können, und *Tools*, die zur grafischen Darstellung (manipulierter) Daten dienen. In der gegenwärtigen experimentellen Testversion gibt es jedoch nur eine beschränkte Anzahl von Tools und Methoden. *AGIL* bietet auch die Möglichkeit, Methoden und Tools selbst zu schreiben und in das System zu integrieren. Dazu wird auf die genannte Grafikbibliothek PHIGS zurückgegriffen. Der Programmierer wird von der Komplexität des PHIGS abgeschirmt, ohne jedoch besondere Einschränkungen in der Leistungsfähigkeit hinnehmen zu müssen: Alle komplizierten Strukturen von PHIGS werden von *AGIL* verwaltet, so daß sich der Programmierer nur noch um die Geometrie der darzustellenden Objekte kümmern muß. Dies ermöglichte, um nur ein Beispiel zu nennen, die testweise Implementierung einer dreidimensionalen Flächendarstellung in weniger als einer Stunde.

Da *AGIL* auf PHIGS beruht, ist seine Leistungfähigkeit stark von der Güte der PHIGS-Implementierung abhängig. Mit der derzeitigen PHIGS-Version auf Sun ist beispielsweise nur eine Grafikausgabe in eine CGM-Datei möglich, wobei jedoch verdeckte Flächen nicht richtig behandelt werden. Um dennoch richtige Bilder von *AGIL* auf Papier zu erhalten, muß daher auf `screendump` zurückgegriffen werden.

Aus diesem Grund kann *AGIL* (zur Zeit) nur empfohlen werden, wenn es darum geht, schnell eine maßgeschneiderte Visualisierungsumgebung, z.B. für ein gewisses Teilgebiet der Algorithmenentwicklung, zu schaffen. Abbildung D.11 auf Seite 125 zeigt ein mit einer *AGIL*-Anwendung erzeugtes Molekül, das, wie oben beschrieben, mit Hilfe von `screendump` erzeugt wurde.

Literatur zu *AGIL*: [73].

2.3.6 MPGS

MPGS[23] ist ein interaktives Programmpaket mit Menu-Steuerung zur Visualisierung von Daten und Ergebnissen aus der Strömungsmechanik und aus der Strukturanalyse.

MPGS wird als verteilte Anwendung zwischen einer UNIX-Workstation und einem CRAY-Supercomputer eingesetzt, wobei der Supercomputer die numerischen Berechnungsschritte und die Workstation die Grafikmanipulationen durchführt.

MPGS ist zugeschnitten auf Anwendungen aus den Bereichen Strömungsmechanik, Strukturanalyse (einschließlich Bruchmechanik), Elektromagnetismus, Wärmeleitung und Erdölexploration.

MPGS kann jedoch auch für viele andere Rendering-Aufgaben, etwa für Meteorologie-Anwendungen, eingesetzt werden.
MPGS kann Datenstrukturen aus

- finiten Elementen,
- finiten Differenzen,
- finiten Volumen und
- Schalenelementen

verarbeiten.
Es bietet Schnittstellen zu den Paketen

- MSC/NASTRAN
- ABAQUS
- PAM-CRASH
- DYNA3D
- RADIOSS
- PATRAN
- FIDAP

und zu Paketen aus dem Bereich der Strömungsberechnung.

2.3.7 UniChem

UniChem[24] ist ein integriertes Programmsystem zur theoretischen Untersuchung von Moleküleigenschaften und der Simulation von chemischen Reaktionen auf molekularem Niveau. Damit erschließen sich neue und/oder einfachere Möglichkeiten bei Untersuchungen chemischer Prozesse, die z.B. in Forschungsprojekten der Chemie, Pharmazie und Pertrochemie durchgeführt werden.

[23] Wir danken Herrn H.-H. Frese für diesen Beitrag.
[24] Wir danken Herrn Dr. T. Steinke für diesen Beitrag.

Zur Nutzung der Visualisierungsmöglichkeiten läuft UniChem als verteilte Anwendung auf einem CRAY Supercomputer und einer UNIX-Workstation. Es bietet eine einheitliche Benutzerschnittstelle zur Datenverwaltung, zum Import und Aufbau von Molekülstrukturen aus Atomen und Fragmenten (aus Fragmentbibliotheken), zur Auswahl des quantenchemischen Berechnungsverfahrens, der Parameterwahl, Informationen über Auftragsstatus und der Visualisierung der Ergebnisse.

Der Kern des UniChem-Systems besteht aus folgenden quantenchemischen Programmen, die einen erheblichen Bedarf an Rechenzeit haben und auf dem Supercomputer CRAY Y-MP2E ablaufen:

- CADPAC, ein Ab-Initio-Programm für quantenchemische Berechnungen von Systemen aus maximal 50 Atomen
- DGauss, ein Dichtefunktional-Programm für Moleküle und Cluster aus maximal 150 Atomen
- MNDO90, ein semi-empirisches Programm (enthält AM1-, PM3-, MNDO- und MINDO/3-Hamilton-Operator) für Moleküle aus maximal 500 Atomen.

Als offenes System gestattet UniChem auch den Zugriff auf weitere Chemie-Programme und die Visualisierung der Ergebnisse auf der Grafik-Workstation. Abbildung D.10 auf Seite 125 zeigt ein mit UniChem erzeugtes Molekül mit seiner Elektronendichte als Bildschirmabzug, Abbildung D.8 wurde mit *Rayshade* aus UniChem-Geometriedaten erzeugt. Mit UniChem erzeugte PostScript-Ausgaben (s. Abb. D.9) erreichen nicht diese Qualität.

Literatur zu UniChem: [64].

2.3.8 Advanced Visualizer (Wavefront)

Der *Advanced Visualizer* von Wavefront ist ein kommerzielles Produkt zur Computer-Animation. Der Advanced Visualizer besteht aus mehreren Moduln, welche die zur Animation notwendigen Aufgaben realisieren:

Model dient zum Erstellen der Geometrie der zu animierenden Objekte. Die Objekte können aus Polygonen, Freiformflächen oder Basisobjekten, wie Würfel, Kugel und dgl., erzeugt werden. In Model werden den Objekten auch die Oberflächeneigenschaften zugewiesen. Model erlaubt sowohl interaktives Arbeiten mit einer grafischen Benutzungsoberfläche, als auch Kommandoeingaben und Scripts. Somit können auch Fremddaten zur Geometrieerzeugung herangezogen werden.

Property Editor ist das Modul zum Generieren von Oberflächeneigenschaften wie Farbe, Reflektion, Glanzlichter, etc. Daneben werden mit dem Property Editor auch Lichtquellen definiert. Die Oberflächeneigenschaften und die Lichtquellen werden von den anderen Moduln verwendet.

Preview ist der Szeneneditor, mit dem die Objekte durch Angabe von Anfangs- und Endpositionen und der Interpolation für dazwischen liegende Positionen animiert werden (*Keyframe Animation*). Es können an die 100 verschiedene Transformationen an den Objekten vorgenommen werden. Diese Manipulationen können auch mit Fremddaten vorgenommen werden, so daß auch durch Simulation gewonnene Daten bei der Animation Verwendung finden können. Nicht nur die in Model erstellten Geometrieobjekte können animiert werden, sondern auch virtuelle Kameras, Lichtquellen, Oberflächeneigenschaften, usw.

Image rendert[25] die Szenen, wie von Preview erstellt, Bild für Bild. Es handelt sich um einen Renderer, der Raytracing und Scanline-Algorithmen verwendet.

Als weiteres Wavefront-Produkt ist am ZIB der *Video Composer* im Einsatz. Der Video Composer ist ein software-basierter interaktiver Videoschnittplatz. Hiermit können Einzelbilder und Bildsequenzen gemischt, überblendet und mit Mitteln der Bildverarbeitung manipuliert werden.[26] Dazu können Titel generiert und animiert werden. Der Video Composer kann Videogeräte direkt steuern, so daß die Ergebnisse des „Compositing“ direkt aufgezeichnet werden können.

Beispiele für mit Wavefront erstellte Bilder sind die sechs Bilder D.12 auf Seite 126 aus dem Vorspann des ZIB-Films, welche mit dem Advanced Visualizer und dem Video Composer erzeugt wurden. Die Abbildung D.13 wurde ebenfalls mit dem Advanced Visualizer erzeugt.

Der Advanced Visualizer und der Video Composer sind am ZIB nur an einer Silicon Graphics Workstation verfügbar.

Literatur zu Advanced Visualizer (Wavefront): s. [51], zu Composer: s. [55] [54].

2.4 Kombinierte Text- und Zeichenprogramme

2.4.1 xfig

Xfig ist ein Menu gesteuerter grafischer Editor, d. h. es ermöglicht dem Benutzer das interaktive Erzeugen und Ändern von grafischen Objekten. Xfig läuft erst ab der X-Version X11R4. Es ist auf SUN und SGI Workstations vorhanden (derzeit jedoch in der Version 2.0 auf SGI und 2.1.6 auf SUN).

Zur Sicherung von Bildern kennt xfig nur ein internes Format. Es verfügt jedoch über viele Dateiausgabeformate; eine Auswahl der wichtigsten Ausgabeformate ist: PostScript, Encapsulated PostScript, LaTeX box und LaTeX picture. Xfig bietet Funktionen zum Erzeugen von Kreisen, Ellipsen, Rechtecken, offenen und geschlossenen Polygonen, Kreisbögen, Text und Splines. Mit einer

[25] s. Rendering im Glossar, Anhang B.

[26] Die Bilder und Sequenzen müssen als Rasterfiles vorliegen. Auf Videogeräten befindliche Sequenzen können hiermit nicht mehr bearbeitet werden; hierzu wäre weitere Hardware (Video-In) erforderlich!

Reihe von Funktionen können Objekte nachträglich verändert werden. Unter anderem können in Polygonen einzelne Punkte eingefügt, gelöscht und verschoben werden. Objekte können skaliert, gedreht, gespiegelt, gefüllt und gruppiert werden.

Xfig kann vorhandene Encapsulated PostScript-Dateien als Ganzes importieren. Dabei wird aber nur die Preview-Bitmap, sofern vorhanden, von xfig angezeigt. Ein solches Objekt kann dann nur noch skaliert und gedreht werden. Ein Bearbeiten einzelner Elemente solch eines importierten Objekts mit xfig-Funktionen ist nicht möglich.
Empfehlung: Zur Weiterverarbeitung von Bildern in LaTeX bzw. TeX sollte die Ausgabe in Encapsulated PostScript erfolgen.

Literatur zu xfig: [62].

2.4.2 Framemaker

FrameMaker ist ein kommerzielles Desktop Publishing Programm. Die zur Zeit am ZIB verfügbare Version ist 3.1X. Framemaker dient dem Erstellen von Dokumenten: von einfachen Notizen über Briefe bis hin zu bebilderten, mehrbändigen Werken. Es unterstützt internationalen, also auch deutschen Zeichensatz. Neben der Textverarbeitung bietet FrameMaker auch die Möglichkeit der Erstellung einfacher Grafiken. Dazu sind folgende Grafikelemente nutzbar:

- Linien in mehreren Stärken, mit oder ohne Pfeil(e) am Ende
- gefüllte Flächen
- Kreisbögen
- Splines
- acht Grundfarben.

Alle Elemente können nachträglich geändert werden, z.B. kann die Steigung an den Kontrollpunkten eines Splines verändert werden, Ecken können gerundet werden, Darstellungsattribute wie Farbe, Linienstärke, Schraffur können neu gesetzt werden, einzelne Punkte eines größeren Elementes können verschoben werden, Vergrößerungen und Verkleinerungen sind möglich. Eine Palette von acht Grundfarben steht zum Einfärben von Grafik- und Textelementen zur Verfügung. Bilder, die im Sun-Rasterformat oder als Encapsulated PostScript vorliegen, können importiert, aber nicht mehr verändert werden. Mit Framemaker verändern lassen sich Bilder (und natürlich Texte) in den Formaten .fm (Framemaker-Format) und .mif (maker interchange format). Für letzteres existieren auch schon Treiber in anderen Programmen, z.B. in GNUPLOT.

Literatur zu Framemaker: s. [26].

2.4.3 Showcase

IRIS Showcase 3.0 ist ein integriertes Textverarbeitungs-, Zeichen- und Präsentationspaket von Silicon Graphics, geeignet auch für Multimedia-Anwendungen. Showcase wird mit dem Betriebssystem für Silicon Graphics Workstations geliefert. Es bietet folgende Möglichkeiten:

- Basistextverarbeitung
- mehrspaltiger Seitenaufbau
- Erstellungen von einfachen Farbgrafiken
- Import von Rasterbildern, 3-D-Bildern und PostScript-Dateien
- Präsentation vorgefertigter Texte und Grafiken (*selfrunning slide show*)
- Erzeugung von Hilfefunktionen für andere Programme
- Einbindung und Abspielen von Audio- und Movie-Files
- Videointegration (Ein- und Ausgabe).
- Hyperscripting(Auslösen von Funktionen, z.B. Audio, Movie,..., durch Mausklick).

Je nach Hardware-Ausbau sind bis zu ca. 16 Milionen Farben (True Colour) für Texte und Grafiken verwendbar. Folgende Grafikelemente sind in Showcase vorhanden:

- Linien in mehreren Stärken, mit oder ohne Pfeil(e) am Ende
- gefüllte Flächen
- Kreisbögen
- Splines.

Alle Elemente können nachträglich geändert werden, z.B. kann die Steigung an den Kontrollpunkten eines Splines verändert werden, Ecken können gerundet werden, Darstellungsattribute wie Farbe, Linienstärke, Schraffur können neu gesetzt werden, einzelne Punkte eines größeren Elementes können verschoben werden, Vergrößerungen und Verkleinerungen sind möglich. Flächen können mit Farbverläufen (Gouraud Shading) gefüllt werden, wenn den Eckpunkten der Flächen unterschiedliche Farben zugewiesen werden. Um 3-D-Bilder zu importieren, müssen sie im Inventor-Format (.iv) (s. Kap. 2.1.7) vorliegen. Somit können auch mit Inventor oder Explorer (s. Kap. 2.3.2) erzeugte Grafiken in Showcase eingebunden werden. Showcase unterstützt auch deutschen Zeichensatz.

2.4.4 XPaint

XPaint ist ein Bitmap Editor zum Manipulieren und Erstellen von Rasterbildern. XPaint liest oder schreibt Rasterbilder im PPM-, XBM- oder XWD-Format. Aufgrund der zahlreichen Konvertierungstools kann jedes beliebige Rasterformat verarbeitet werden (siehe Kap. 3.1.1). XPaint ist ein public domain Programm auf der Basis von X11. Es bietet folgende Grafik-Primitive zum Zeichnen an:

- Linien in mehreren Stärken
- Rechtecke, auch gefüllt
- offenen und geschlossene Polygone, gefüllt oder nicht
- Ellipsen
- Kreisbögen
- Text, in vielen Fonts und Größen.

Diese Primitive können nach dem Malen nicht mehr als ein Objekt manipuliert werden. Die letzte Aktion kann noch durch ein "Undo" rückgängig gemacht werden. Löschen und Kopieren wird ansonsten durch Aufziehen eines Rechteckes über einem Teil der Malfläche oder durch Umranden mit einem geschlossenen Linienzug durchgeführt. Darüberhinaus gibt es keine Möglichkeit vorhandene Bildelemente zu manipulieren, es bleibt nur das Übermalen durch ein anderes Primitiv. Zum Füllen von Flächen stehen die acht Grundfarben und verschiedene Füllmuster (Pattern) zur Verfügung. Die Füllmuster können um eigene Muster ergänzt werden.

Spray- und Paint-Brush-Funktionen sind ebenfalls vorhanden. Es können mehrere Fenster zum Malen (Canvas) eröffnet werden, wobei auch von einem zum anderen Fenster kopiert werden kann (Cut and Paste).

2.4.5 CLRpaint

CLRpaint ist ein Bitmap Editor zum Manipulieren und Erstellen von Rasterbildern. CLRpaint liest oder schreibt Rasterbilder im Silicon Graphics RGB-Format, Targa-, GIF-, TIFF-, JPEG- und Kodak Photo-CD-Format. Aufgrund der zahlreichen Konvertierungstools kann jedes beliebige Rasterformat verarbeitet werden (siehe Kap. 3.1.1). CLRpaint ist ein public domain Programm auf der Basis der SGI GL. Es bietet im wesentlichen auch die gleiche Funktionalität wie XPaint (siehe Kap. 2.4.4), verfügt aber darüberhinaus noch über bildverarbeitende Funktionen.

Empfehlung: Durch Nutzung der SGI GL ergibt sich eine gegenüber XPaint wesentlich verbesserte Darstellung, so daß CLRpaint XPaint, wenn irgend möglich, vorzuziehen ist.

Literatur zu CLRpaint: s. [7].

2.4.6 TEX und LATEX

Zumindest für mathematisch orientierte Publikationen hat sich neben TEX insbesondere die darauf aufbauende Macrosprache LATEX [34] als De-Facto-Standard-Textverarbeitungssystem etabliert. Im Zusammenhang mit Grafik werden zweierlei Anforderungen an TEX gestellt: zum einen die Erstellung von Bildern selbst und das Einfügen von Bildern, die auf anderem Wege erzeugt wurden.

Der erste Aspekt wird von LATEX mit der `picture`-Umgebung oder dem Makropaket PICTEX [70] realisiert. Die Grafikfähigkeiten von LATEX sind eine Teilmenge derer von PICTEX. Mit speziellen Makros können einfache Liniengrafiken oder mit PICTEX auch Funktionsdarstellungen produziert werden. Der Vorteil einer solchen Integration von Grafik in ein TEX-Dokument besteht in der Portabilität innerhalb der TEX-Welt. Andererseits ist solch eine Bilderstellung mit einer Beschreibungssprache wesentlich unhandlicher als eine interaktive Gestaltung am Bildschirm. An diesem Punkt kann jedoch auf *xfig* zurückgegriffen werden (vgl. Kap. 2.4.1), mit dem interaktiv erstellte Grafiken auch im LATEX-Format ausgegeben werden können. Als Problem hat sich dabei jedoch die eingeschränkte Zeichensatzauswahl erwiesen.

Zur Einbeziehung von separat erstellten Bildern bietet TEX keinen standardisierten Weg. Es muß auf das `\special`-Makro zurückgegriffen werden, dessen Wirkung von dem jeweils verwendeten DVI-Treiber abhängig ist. Für den Ausdruck auf PostScript-Druckern (vgl. Kap. 4.2.2) steht am ZIB das Programm `dvips` zur Verfügung, das wahlweise auch eine PostScript-Datei erzeugen kann. Es arbeitet mit dem `.sty`-File `psfig.sty` zusammen, das eine einfache Einbindung von Encapsulated PostScript (`.eps`) Bildern bietet (vgl. Kap. 3.1.3). Dazu muß als `\documentstyle` zusätzlich `psfig` angegeben werden. Anschließend können im Text beliebige `.eps`-Bilder mit dem Befehl `\psfig{figure=<filename>}` eingebunden werden. Es ist auch möglich, die Höhe und Breite des Bildes vorzugeben oder gar einen Drehwinkel. Alle Grafiken in diesem Handbuch wurden auf diesem Weg eingefügt.

Oft besteht der Wunsch, in PostScript-Grafiken an einer bestimmten Stelle LATEX-Text zu positionieren. Das ermöglichen das LATEX-Macro `psfrag` und das Shellscript `\ps2psfrag`. Mit dem Shellscript `ps2psfrag` wird das PostScript-File vorverarbeitet und ein File .psfrag angelegt, das alle im PostScript-File enthaltene Textfragemente und deren Positionen enthält. Damit der zu ersetzende Text in dieser Vorverarbeitung gefunden wird, muß er als Argument einer `show` Anweisung im PostScript-File vorkommen, d.h., er darf nicht über mehrere Anweisungen gesplittet sein. Die Ersetzung wird mit dem LATEX-Macro `\psfrag` keyword latex-Text ausgeführt. Es überschreibt das Keyword mit dem angegebenen LATEX-Text in der LATEX-Ausgabe. Das PostScript-File muß mit `\epsfbox` in die LATEX-Quelle eingebunden werden. Eine zweite Möglichkeit ist, LATEX-Text direkt in das PostScript-File zu schreiben, und zwar in der Form `\tex` latex-Text. Damit entfällt die Ersetzung mittels `psfrag`.

Das `psfrag`-Makro wird durch das Einbinden des psfrag-Stylefiles zur Verfügung gestellt. Die `psfrag`- und `epsf`-Option müssen bei der Festlegung des Do-

kumentes `\dokumentstyle` angegeben werden.

Literatur zu TEX: [70], zu LATEX: [34] zu GNUPLOT mit LATEX: [36].

2.5 Renderer

2.5.1 Rayshade

Rayshade ist ein public-domain-Programm zur Erzeugung von fotorealistischen Bildern. Die Bilder werden mittels Raytracing (Strahlverfolgung) berechnet. Rayshade liest eine Szenenbeschreibung ein und erzeugt daraus ein farbiges Rasterbild. Eine Szenenbeschreibung für Rayshade besteht aus einer Menge von Objekten und deren Attributen, Lichtquellen, Kamerastandpunkt und -einstellungen. Basisobjekte sind beispielsweise

- Kugeln
- Zylinder
- Kästen
- Ebenen
- Dreiecke
- Polygone.

Diese Basisobjekte können zu komplexeren Objekten zusammengesetzt werden. Aus Objekten können wiederum andere Objekte durch Anwendung von Methoden der *Constructive Solid Geometry* (CSG), d.h. durch Vereinigungs-, Schnitt- oder Differenzmengenbildung von Objekten, gebildet werden. Die Erzeugung von Stereobildern wird unterstützt durch die Möglichkeit, den Augenabstand anzugeben.
Ein einfaches Beispiel einer Szenenbeschreibung ist

```
sphere 2 0 0 0,
```

die eine Kugel mit dem Radius 2 und dem Mittelpunkt an der Position (0,0,0) im Raum beschreibt. Für Betrachterstandpunkt, Kameraeinstellungen und Beleuchtung werden Defaultwerte eingesetzt.

Das Berechnen eines Bildes mit Rayshade kann u. U. sehr lange dauern (Dauer nach oben offen!). Die Berechnungsdauer hängt von der Komplexität der Szene, der gewünschten Bildgröße und der eingestellten Bildqualität (etwa Tiefe der Strahlverfolgung) ab. Deshalb sollte man ein Bild zunächst mit niedriger Qualität und in geringer Auflösung testweise erzeugen (Preview-Qualität) und erst, wenn man mit den gewählten Einstellungen zufrieden ist, das hochqualitative, hochaufgelöste Bild herstellen.

Rayshade erzeugt Rasterbilder im RLE-Format des Utah Raster Toolkits (s. Kap. 3.2.6). Diese Bilder kann man am Bildschirm ausgeben und auch nach Colour-PostScript wandeln (siehe Kap. 3.2.6 und 3.1.3). Die Abbildung D.14 (S.

127) zeigt eine mit Rayshade gerenderte[27] Szene. Die Schrift wurde nachträglich mit Raster Tools eingefügt. Auch die Abb. D.8 (S. 125) wurde mit Rayshade gerendert. Die Eingabedaten für Rayshade wurden in diesem Fall mithilfe eines eigenen Programms aus Unichem-Output erstellt.

Literatur zu rayshade: s. [33].

2.5.2 Bob

Bob (**B**rick **o**f **b**ytes) ist ein Volume-Rendering-Programm zur schnellen Transparenzdarstellung von Volumendaten. Die Volumendaten müssen als reguläres kartesisches dreidimensionales Gitter vorliegen. Die darin befindlichen Gitterzellen werden *Voxel* (Volumen Element) in Analogie zu *Pixel* genannt. Jedes Voxel hat eine Farbe und einen Alphawert[28] (Transparenzwert). Zur Darstellung wird das Volumen in Blickrichtung von hinten nach vorn gezeichnet, wobei ein Voxel als ein Polygon mit einer Farbe und einem Alphawert ausgegeben wird. Der Alphawert bestimmt, wie der Farbwert mit Farbwerten schon dargestellter Voxel gemischt werden soll. Auf diese Weise kann ein Volumen als Ganzes dargestellt werden, ohne Strukturen im Inneren des Volumens zu verdecken.

Bob's Datenformat ist sehr einfach: Pro Voxel wird ein Byte abgelegt. Jedes Voxel ist ein Index zu einer Farbtabelle und gleichzeitig ein Index zu einer Tabelle mit Alphawerten. Die Voxeldaten werden binär ohne Header abgelegt. Die Reihenfolge ist so, daß die Voxel zunächst entlang der X-Achse, dann in Y-Richtung in Ebenen und dann entlang der Z-Achse eingelesen werden.

Zur schnellen Darstellung benutzt Bob Alpha-Bitplanes[29], falls vorhanden. Akzeptable Ergebnisse in erträglicher Zeit sind im ZIB daher nur auf der SGI-Workstation `grafs11` erzielbar. Auf den Indigos wird das Alpha-Blending per Software emuliert, so daß die Erzeugung eines Bildes erheblich länger dauert. Die Personal Iris' bieten keine Hard- oder Software-Unterstützung für Alpha-Blending, daher sind mit Bob dort keine transparenten Darstellungen möglich. Es wird immer nur das Voxel mit dem höchsten Alphawert entlang der Blickrichtung dargestellt.

Bob bietet die Möglichkeit, einen Ausschnitt (Quader) aus dem Volumen zu visualisieren. Damit kann bei großen Datensätzen die Rendering-Zeit in Grenzen gehalten werden. Das zu betrachtende Volumen kann mit der Maus im Darstellungsfenster gedreht, verschoben und vergrößert werden. Eine Animationsunterstützung erlaubt es, das Volumen zu drehen und die erzielten Einzelbilder abzuspeichern.

Zur interaktiven Erstellung von Farbtabellen und Alphatabellen steht das Programm *icol* zur Verfügung. Bob und icol kommunizieren miteinander, so daß eine Veränderung der Farbtabelle sofort eine veränderte Darstellung in Bob zur Folge hat. Das gleiche gilt für die Alphatabellen.

Es sei nicht verschwiegen, daß man mit Bob, wie überhaupt mit der hier verwendeten Art des Volume-Rendering, keine "schönen" Bilder erhält. Solche

[27] s. Rendering im Glossar, Anhang B.
[28] s. Alpha-Kanal im Glossar, Anhang B.
[29] s. Bit-Ebene im Glossar, Anhang B.

erzielt man nur mit aufwendigeren, heute noch rein software-basierten Methoden. Die Abbildung D.15 auf Seite 127 zeigt eine mit Bob berechnete Transparenzdarstellung eines Schädels mit eingeblendeten Temperaturfeld. Durch die Verwendung einer problemangepaßten Farbtabelle kann das erwärmte Gebiet optisch deutlich hervorgehoben werden.

Literatur zu Bob: [9].

A picture is an intermediate something between a thought and a thing.

– Samuel Taylor Coleridge

Kapitel 3

Speicherung und Bearbeitung von Bildern

Beim Austausch von Grafiken zwischen verschiedenen Applikationen oder Hardware-Plattformen ergibt sich für den Benutzer fast zwangsläufig die Notwendigkeit, mit verschiedenen Bildformaten vertraut zu sein und Möglichkeiten der Konvertierung zu kennen. Bei der Ausgabe von Grafiken, etwa auf einen Diabelichter oder Farbdrucker, ist es oft unumgänglich, die Bilder einer Nachbearbeitung zu unterziehen, um ein ansprechendes Resultat zu erhalten. Diskretisierungsartefakte müssen ausgebügelt, Helligkeit und Kontrast angepaßt werden.

In diesem Kapitel wird ein Überblick über die wichtigsten, heute verwendeten Grafikformate gegeben, wobei der Aspekt der Konvertierung verschiedener Formate untereinander im Vordergrund steht. Zudem sollen Möglichkeiten der Nachbearbeitung, insbesondere von Rasterbildern, dargestellt werden.

3.1 Bildformate und Format-Konvertierung

Grundsätzlich unterscheidet man zwischen *Rasterformaten*, *Vektorformaten* und *Seitenbeschreibungssprachen*. Vektorformate sind in erster Linie für Liniengrafiken bestimmt. Sie sind das geeignete Mittel zur Ansteuerung von Stiftplottern. Auch Schriften werden heute oft über die Buchstabenumrisse in Vektorform gespeichert. Mit Rasterformaten lassen sich dagegen ganz beliebige Formen und Farbverläufe kodieren. Der große Nachteil von Rasterformaten ist aber, daß sie nicht geräteunabhängig sind. Ein Rasterbild, das einen großen Teil des Bildschirms ausfüllt, kann auf einem anderen Ausgabegerät, wie zum Beispiel einem höherauflösenden Diabelichter, winzig klein erscheinen. Man kann das Rasterbild zwar vergrössern, aber dadurch wird die Pixelstruktur sichtbar, oder das Bild erscheint unscharf.

Ein Vektorformat kann man immer noch in ein Rasterformat wandeln, aber umgekehrt ist es nur sehr eingeschränkt möglich. Wenn also die Vektorinformation zur Verfügung steht, sollte auch ein entsprechendes Format verwendet werden. Viele CAD- und Zeichenprogramme haben eigene Vektorformate.

Seitenbeschreibungssprachen verbinden die Vorteile von Vektor- und Rasterformaten. Die bei weitem am meisten verwendete Seitenbeschreibungssprache ist PostScript. PostScript ist eigentlich eine richtige Programmiersprache und kein einfaches Bildformat mehr. Der Nachteil bei der Verwendung mächtiger

Sprachen wie PostScript besteht darin, daß man aufwendige Interpreter zum Darstellen der Bilder braucht. So unterstützen die meisten Anwendungsprogramme heute PostScript-Ausgabe, es gibt aber kein einziges, das beliebige PostScript-Dateien auch wieder einlesen und auf PostScript-Ebene uneingeschränkt weiterbearbeiten kann.

3.1.1 Rasterformate

Rasterbilder werden viel verwendet, weil sie problemlos auf dem Bildschirm angezeigt und von diesem kopiert werden können und weil viele Visualisierungstechniken ausschließlich Rasterbilder erzeugen können. Sobald in einem Bild schattierte Farbverläufe auftreten, sind Vektorformate ungeeignet, denn die Bildberechnungstechniken führen zu Rasterbildern.

Im folgenden sollen häufig verwendete Rasterformate vorgestellt werden. Neben einer kurzen Formatbeschreibung werden auch nützliche Hilfsprogramme erwähnt, mit denen das jeweilige Format erzeugt oder bearbeitet werden kann. Soweit spezifische Probleme mit einem Format bekannt sind, werden diese angeführt. Zu jedem Format gibt es eine Liste, in der Konvertierungsmöglichkeiten angegeben sind. Diese Liste kann jedoch in keinem Fall vollständig sein. Es gibt noch Dutzende weiterer Rasterformate, für die unter Umständen auch Konvertierungstools am ZIB zur Verfügung stehen. Wir haben uns hier absichtlich auf die Grafikformate beschränkt, die im Workstationbereich am häufigsten Anwendung finden.

Die meisten der im folgenden erwähnten Konvertierungstools sind Bestandteil größerer Pakete zur Verarbeitung von Rasterbildern, sogenannter *Raster Toolkits* (s. Kap. 3.2). Auf diese Pakete wird weiter unten noch genauer eingegangen. Wir wollen nur kurz angeben, welches Konvertierungstool zu welchem Toolkit gehört.

Konvertierungsprogramm	Raster Toolkit
convert	Image Magick
imconv	SDSC Image Tools
fbcat	FBM Fuzzy Pixmap Library
fromxyz, toxyz	Iris Raster Tools
pbmtoxyz, xyztopbm	PBMPLUS Package
pgmtoxyz, xyztopgm	PBMPLUS Package
ppmtoxyz, xyztoppm	PBMPLUS Package
rletoxyz, xyztorle	Utah Raster Toolkit

Einige Konvertierungstools gehören zu keinem der Raster Toolkits. Es handelt sich dabei um

- **gs**, einen frei verfügbaren PostScript-Interpreter mit der Möglichkeit, Rasterfiles zu erzeugen
- **img_export** und **img_import**, Konvertierungsprogramme, die zum Visualisierungspaket **apE** gehören
- **xwd2ps**, ein Programm zur Wandlung von X-Window-Dumps nach Encapsulated PostScript.

Encapsulated PostScript (eps)

Alternative Endungen: .epi, .epsi, .epsf

Bemerkungen: *Encapsulated PostScript* wird überwiegend dazu verwendet, Bilder und Grafiken in andere Dokumente einzubinden. Von gewöhnlichem PostScript unterscheidet sich *Encapsulated PostScript* im wesentlichen durch die zusätzliche Angabe einer *bounding box*. Eine genauere Beschreibung hierzu befindet sich im Kapitel 3.1.3 auf Seite 57.

Formatspezifikation: Alle gängigen Kodierungen wie Bitmaps, Graustufenbilder und Farbbilder mit 8-Bit *pseudo color* oder 24-Bit *true color* sind möglich. Da PostScript eine Programmiersprache ist, lassen sich auch beliebige andere Kodierungsformen und Kompressionsschemata realisieren.

Probleme: Bei der Umwandlung einer Rastergrafik nach PostScript verbessert sich die Qualität der Darstellung nicht. Die Daten liegen weiterhin in Rasterform vor und werden nicht etwa in ein Vektorformat umgewandelt.
Obwohl man mit Interpretern wie **gs** Rastergrafiken erzeugen kann, unterscheidet sich dieser Vorgang dennoch deutlich von der Konvertierung zwischen reinen Rasterformaten. Die Pixeldaten in PostScript werden notwendigerweise einer Transformation unterzogen (Skalierung oder Verschiebung auf einem virtuellen Blatt Papier). Eine nach PostScript und wieder zurück gewandelte Rastergrafik wird sich deshalb im allgemeinen schon in der Größe vom Original unterscheiden.

Konvertierungstools:

Format		... to eps	eps to ...
gif	CompuServe GIF format	imconv, convert	-
hdf	Hierarchical Data Format	imconv	-
mpnt	Apple MacPaint file	imconv	-
pbm	Portable Bitmap	imconv, convert	gs
pcx	IBM PC Paintbrush file	imconv	-
pgm	Portable Graymap	imconv, convert	gs
pic	Pixar Picture file	imconv, convert	-
pict	Apple QuickDraw file	imconv	-
ppm	Portable Pixmap	imconv, convert ppmtops	gs
ps	PostScript	-	- [a]
rgb	Silicon Graphics RGB	imconv, tops	gs [b]
ras	Sun Rasterfile	imconv, convert	-
rla	Wavefront image file	imconv	-
rle	Utah RLE image file	imconv, convert	-
tiff	Tagged Image File	imconv, convert	-
tga	Targa image file	imconv	gs [b]
yuv	Abekas YUV image file	-	-
xbm	X11 Bitmap file	imconv, convert	-
xwd	X11 Window Dump	imconv, convert, xwd2ps	-

[a] *Encapsulated PostScript* ist zugleich auch *PostScript*
[b] Nur auf SGI-Maschinen

PBMPLUS Toolkit Formate (pbm, pgm, ppm)

Alternative Endungen: keine

Bemerkungen: Diese Formate werden von Jef Poskanzers PBMPLUS Toolkit verwendet. Die Abküzungen stehen für *portable bitmap*, *portable graymap* und *portable pixmap*.

Formatspezifikation: 1 Bit monochrom (*pbm*), 8 Bit Graustufen (*pgm*) und 24 Bit RGB-Farbe (*ppm*) sind möglich. Die Daten können wahlweise binär oder in ASCII Form vorliegen. Weitere Optionen gibt es nicht. Deshalb gelten diese Formate als sehr robust.

Probleme: nicht bekannt

Konvertierungstools:

Format		... to pnm	pnm to ...
eps	Encapsulated PostScript	-	imconv, convert ppmtops
flx	apE image file	-	-
gif	CompuServe GIF format	imconv, convert, fbcat, giftoppm	imconv, convert, fbcat, ppmtogif
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv, macptopbm	imconv, pbmtomacp
pcx	IBM PC Paintbrush file	imconv, fbcat, ppmtopcx	imconv, fbcat, pcxtoppm
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv, picttoppm	imconv, ppmtopict
ps	PostScript	-	imconv, convert ppmtops
ras	Sun Rasterfile	imconv, convert, fbcat, rasttopnm	imconv, convert, fbcat, pnmtorast
rgb	Silicon Graphics RGB	imconv, toppm	imconv, fromppm
rla	Wavefront image file	imconv	imconv
rle	Utah RLE image file	imconv, convert, rletoppm	imconv, convert, ppmtorle
tiff	Tagged Image File	imconv, convert, tifftopnm	imconv, convert, pnmtotiff
tga	Targa image file	imconv, tgatoppm	imconv, ppmtotga
yuv	Abekas YUV image file	-	-
xbm	X11 Bitmap file	imconv, convert, xbmtopbm	imconv, convert, pbmtoxbm
xwd	X11 Window Dump	imconv, convert, xwdtopnm	imconv, convert, pnmtoxwd

Sun Rasterfile (ras)

Alternative Endungen: .scr, .sr, .sun

Bemerkungen: Dieses Format ist das offizielle Rasterformat der Firma Sun Microsystems Inc. Viele Tools auf Workstations dieses Herstellers arbeiten damit, so zum Beispiel **screendump** und **screenload**.

Formatspezifikation: *Sun Rasterfiles* unterstützen Farbtiefen von 1, 8, 24 und 32 Bit. Alle Tiefen können optional eine Farbtabelle enthalten. In der Regel ist das jedoch nur bei 8 Bit Farbtiefe der Fall. Run-length encoding, bei Sun *byte-encoding* genannt, ist möglich.

Probleme: nicht bekannt

Konvertierungstools:

Format		... to ras	ras to ...
eps	Encapsulated PostScript	-	imconv, convert
flx	apE image file	img_export	img_import
gif	CompuServe GIF format	imconv, convert, fbcat	imconv, convert, fbcat
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv	imconv
pbm	Portable Bitmap	imconv, convert, fbcat, pnmtorast	imconv, convert, fbcat, rasttopnm
pcx	IBM PC Paintbrush file	imconv, fbcat	imconv, fbcat
pgm	Portable Graymap	imconv, convert, pnmtorast	imconv, convert, rasttopnm
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv	imconv
ppm	Portable Pixmap	imconv, convert, pnmtorast	imconv, convert, rasttopnm
ps	PostScript	-	imconv
rgb	Silicon Graphics RGB	imconv, tosun	imconv, fromsun
rla	Wavefront image file	imconv	imconv
rle	Utah RLE image file	imconv, convert rletoras	imconv, convert rastorle
tiff	Tagged Image File	imconv, convert	imconv, convert
tga	Targa image file	imconv	imconv
yuv	Abekas YUV image file	- [a]	- [a]
xbm	X11 Bitmap file	imconv, convert	imconv, convert
xwd	X11 Window Dump	imconv, convert	imconv, convert

[a] Konvertierung über *rgb* als Zwischenformat möglich.

Silicon Graphics RGB Image File (rgb)

Alternative Endungen: .sgi, .iris

Bemerkungen: Dieses Format ist das offizielle Rasterformat der Firma Silicon Graphics Inc. Es gibt ein eigenes Raster Toolkit für dieses Format. Mit **snapshot** läßt sich ein Bildschirmausschnitt als *RGB-File* speichern. Mit **ipaste** kann man ein *RGB-File* auf dem Bildschirm darstellen. **imgworks** und **enhance** sind nützliche Utilities zur Bildverarbeitung. **istat** zeigt die Bildgröße und andere Informationen an. *RGB-Files* lassen sich weiterhin in das Textsystem **showcase** einbinden.

Formatspezifikation: *RGB-Files* sind in der Regel RGB-kodiert mit 24-Bit Farbtiefe. Run-length encoding wird unterstützt. Graustufenbilder, Bitmaps und optionaler Alpha channel sind möglich, aber unüblich.

Probleme: Die SDSC Image Tools (**imconv**, **imgray** usw.) unterstützen nur 24-Bit Bilder. Graustufenbilder werden falsch interpretiert.

Konvertierungstools:

Format		... to rgb	rgb to ...
eps	Encapsulated PostScript	gs [a]	imconv
flx	apE image file	- [b]	- [b]
gif	CompuServe GIF format	imconv	imconv
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv, frommac	imconv, tomac
pbm	Portable Bitmap	imconv	imconv
pcx	IBM PC Paintbrush file	imconv	imconv
pgm	Portable Graymap	imconv	imconv
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv	imconv, topict
ppm	Portable Pixmap	imconv, fromppm	imconv, toppm
ps	PostScript	gs [a]	imconv, tops
ras	Sun Rasterfile	imconv, fromsun	imconv, tosun
rla	Wavefront image file	imconv, fromrla	imconv, torla
rle	Utah RLE image file	imconv, rletoiris	imconv, iristorle
tiff	Tagged Image File	imconv	imconv
tga	Targa image file	imconv, fromtarga	imconv, totarga
yuv	Abekas YUV image file	fromyuv	toyuv
xbm	X11 Bitmap file	imconv, fromxbm	imconv
xwd	X11 Window Dump	imconv	imconv

[a] Ghostscript mit speziellem Treiber, nur auf SGI-Maschinen lauffähig.

[b] Konvertierung über *pic* (Pixar) als Zwischenformat möglich.

Utah Run-Length encoded image file (rle)

Alternative Endungen: keine

Bemerkungen: Dieses Format wird im *Utah Raster Toolkit* benuzt.

Formatspezifikation: Das *Utah RLE Format* unterstützt alle gängigen Farbtiefen von 1, 8 und 24 Bit ebenso wie Alphawerte und optionale Farbtabelle. Außerdem gibt es eine Reihe von zusätzlichen Features, wie das Einbinden von Kommentaren, Angabe absoluter Positionen oder die Möglichkeit, mehrere Bilder in einem File zu halten.

Probleme: *Utah-RLE*-Besonderheiten wie Kommentare, absolute Positionen oder mehrere Bilder in einem File werden von den meisten Konvertierprogrammen ignoriert.

Konvertierungstools:

Format		... to rle	rle to ...
eps	Encapsulated PostScript	-	imconv, rletops
flx	apE image file	img_export	img_import
gif	CompuServe GIF format	imconv, convert giftorle	imconv, convert rletogif
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv, rletopaint	imconv, panttorle
pbm	Portable Bitmap	imconv, convert	imconv, convert
pcx	IBM PC Paintbrush file	imconv	imconv
pgm	Portable Graymap	imconv, convert pgmtorle	imconv, convert
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv	imconv
ppm	Portable Pixmap	imconv, convert, ppmtorle	imconv, convert, rletoppm
ps	PostScript	-	imconv, rletops
ras	Sun Rasterfile	imconv, convert, rasttorle	imconv, convert, rletorast
rgb	Silicon Graphics RGB	imconv, iristorle	imconv, rletoiris
rla	Wavefront image file	imconv, rlatorle	imconv, rletorla
tiff	Tagged Image File	imconv, convert, tifftorle	imconv, convert, rletotiff
tga	Targa image file	imconv	imconv
yuv	Abekas YUV image file	- [a]	- [b]
xbm	X11 Bitmap file	imconv, convert	imconv, convert
xwd	X11 Window Dump	imconv, convert	imconv, convert

[a] Konvertierung über *rgb* als Zwischenformat möglich.

Targa TrueVision image file (tga)

Alternative Endungen: keine

Bemerkungen: Das Targa TrueVision Bildformat stammt von AT&T und wird überwiegend im PC-Bereich verwendet. Am ZIB ist es relevant für den Diabelichter. Dieses Gerät kann direkt nur Bilder darstellen, die im Targa-Format vorliegen.

Formatspezifikation: Es gibt eine ganze Reihe von Optionen, die von der Unterstützung aller denkbaren Farbtiefen und Farbtabellen bis hin zur Verwendung komplexer Kompressionsalgorithmen reichen (es sind Differenzen-Kodierung und LZW-Kompression möglich). In der Praxis wird aber fast ausschließlich ein "aufgeblasener" Formattyp mit 32 Bit pro Pixel verwendet, oder seltener auch eine Variante mit Run-Length Encoding.

Probleme: nicht bekannt

Konvertierungstools:

Format		... to tga	tga to ...
eps	Encapsulated PostScript	gs	imconv
flx	apE image file	img_export	img_import
gif	CompuServe GIF format	imconv	imconv
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv	imconv
pbm	Portable Bitmap	imconv, ppmtotga	imconv, tgatoppm
pcx	IBM PC Paintbrush file	imconv	imconv
pgm	Portable Graymap	imconv	imconv
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv	imconv
ppm	Portable Pixmap	imconv, ppmtotga	imconv, tgatoppm
rgb	Silicon Graphics RGB	imconv, totarga	imconv, fromtarga
ras	Sun Rasterfile	imconv	imconv
rla	Wavefront image file	imconv	imconv
rle	Utah RLE image file	imconv	imconv
tiff	Tagged Image File	imconv	imconv
yuv	Abekas YUV image file	- [a]	- [a]
xbm	X11 Bitmap file	imconv	imconv
xwd	X11 Window Dump	imconv	imconv

[a] Konvertierung über *rgb* als Zwischenformat möglich.

X11 Bitmap Image (xbm)

Alternative Endungen: .bm

Bemerkungen: *X11 Bitmap Images* werden von dem Iconeditor **bitmap** des MIT X11 Window Systems generiert. Die meisten Tools von X11 benutzen dieses Format, um Cursorformen, Icons oder andere monochrome Symbole zu definieren.

Formatspezifikation: Reines Bitmap-Format. Zusätzlich kann ein Pixel ausgezeichnet werden, der sogenannte *hot spot*. Dies ist für Cursorsymbole sinnvoll.

Probleme: nicht bekannt

Konvertierungstools:

Format		... to xbm	xbm to ...
eps	Encapsulated PostScript	-	imconv, convert
flx	apE image file	- [a]	- [a]
gif	CompuServe GIF format	imconv, convert	imconv, convert
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv	imconv
pbm	Portable Bitmap	imconv, convert pbmtoxbm	imconv, convert xbmtopbm
pcx	IBM PC Paintbrush file	imconv	imconv
pgm	Portable Graymap	imconv, convert	imconv, convert
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv	imconv
ppm	Portable Pixmap	imconv, convert	imconv, convert
ps	PostScript	-	imconv, convert
rgb	Silicon Graphics RGB	imconv	imconv, fromxbm
ras	Sun Rasterfile	imconv, convert	imconv, convert
rla	Wavefront image file	imconv	imconv
rle	Utah RLE image file	imconv, convert	imconv, convert
tiff	Tagged Image File	imconv, convert	imconv, convert
tga	Targa image file	imconv	imconv
yuv	Abekas YUV image file	- [b]	- [b]
xwd	X11 Window Dump	imconv, convert	imconv, convert

[a] Konvertierung über *pic* (Pixar) als Zwischenformat möglich.

[b] Konvertierung über *rgb* als Zwischenformat möglich.

X11 Window Dump (xwd)

Alternative Endungen: .x11

Bemerkungen: Das *X11 Window Dump* Format wird von den **xwd**, **xwud** und **xpr** Tools des MIT X11 Window Systems benutzt. Mit diesen Tools können Fenster vom Bildschirm in ein File kopiert werden, Files wieder auf dem Bildschirm dargestellt bzw. auf einem Printer ausgegeben werden.

Formatspezifikation: Es werden Farbtiefen von 8 und 24 Bit unterstützt. Farbtabellen sind möglich.

Probleme: Man erhält des öfteren falsche Farben, insbesondere, wenn Farbtabellen verwendet werden. Bei der Bildschirmdarstellung mit **xwud** kann das evtl. durch die Wahl einer anderen *visual class* mit der *-vis* Option korrigiert werden.

Konvertierungstools:

Format		... to xwd	xwd to ...
xwd	X11 Window Dump	imconv, convert	imconv, convert
eps	Encapsulated PostScript	-	imconv, convert, xwd2ps
flx	apE image file	- [a]	- [a]
gif	CompuServe GIF format	imconv, convert	imconv, convert
hdf	Hierarchical Data Format	imconv	imconv
mpnt	Apple MacPaint file	imconv	imconv
pbm	Portable Bitmap	imconv, convert, pnmtoxwd	imconv, convert, xwdtopnm
pcx	IBM PC Paintbrush file	imconv	imconv
pgm	Portable Graymap	imconv, convert, pnmtoxwd	imconv, convert, xwdtopnm
pic	Pixar Picture file	imconv	imconv
pict	Apple QuickDraw file	imconv	imconv
ppm	Portable Pixmap	imconv, convert, pnmtoxwd	imconv, convert, xwdtopnm
ps	PostScript	-	imconv, convert, xwd2ps
rgb	Silicon Graphics RGB	imconv	imconv, fromxwd
ras	Sun Rasterfile	imconv, convert	imconv, convert
rla	Wavefront image file	imconv	imconv
rle	Utah RLE image file	imconv, convert	imconv, convert
tiff	Tagged Image File	imconv, convert	imconv, convert
tga	Targa image file	imconv	imconv
yuv	Abekas YUV image file	- [b]	- [b]
xbm	X11 Bitmap file	imconv, convert	imconv, convert

[a] Konvertierung über *pic* (Pixar) als Zwischenformat möglich.
[b] Konvertierung über *rgb* als Zwischenformat möglich.

3.1.2 Vektorformate

Die derzeit am ZIB verwendeten Vektorgrafikformate zur Speicherung von Bildern sind GKS Metafile (GKSM), CGM (**C**omputer **G**raphics **M**etafile) und HPGL (**H**ewlett-**P**ackard **G**raphics **L**anguage). Alle drei Formate sind 2-D-Formate, d.h. Anwendungsprogramme, die mit 3-D-Daten arbeiten, müssen diese in 2-D wandeln, ehe sie in einem dieser Formate abspeichern können.

GKSM - GKS Metafile

Allgemein verwendete Endungen: .gksm, .gksm*nn*

Bemerkungen: Ein GKSM ist streng betrachtet kein statisches Bild, sondern eine Art Sitzungsprotokoll, in dem alle grafischen Ausgaben, die an das Ausgabegerät[1] gehen, aufgezeichnet werden. Somit werden auch evtl. dynamische Bildänderungen mit aufgezeichnet.

Probleme: In der GKS-Norm wird nur gesagt, daß es eine Aus- und Eingabe von Bilddateien gibt, und die prozedurale Schnittstelle dazu ist beschrieben; das eigentliche Format der Bilddatei ist aber nicht Teil der Norm. Damit sollte einer geplanten Normung eines Bilddateiformates nicht vorgegriffen werden. Es gibt einen *Anhang* zur Norm, in dem ein GKSM definiert wird; dieser ist jedoch nicht Teil der Norm, so daß bei unterschiedlichen GKS-Implementierungen die Metafiles fast immer unterschiedlich und somit nicht übertragbar sind.

Konvertierungstools: Jede GKS-Implementierung kann GKS-Metafiles interpretieren und auf alle anderen Ausgabegeräte dieser Implementierung ausgeben.

GKSM-Erzeuger sind folgende Programme, bzw. Grafikbibliotheken:

- FUGKS erzeugt und interpretiert Metafiles in mehreren Formaten.
- XGKS
- GRAZIL benutzt als Basisgrafikbibliothek GKS und kann somit Metafiles erzeugen.

CGM - Computer Graphics Metafile

Allgemein verwendete Endungen: .cgm, .ccgm, .tcgm

Bemerkungen: CGM ist ein international genormtes Format (ISO 8632) zur Speicherung von statischen Bildern.

[1] in der GKS-Philosophie ist ein Metafile ebenso ein Ausgabegerät (Workstation) wie ein Bildschirm.

Formatspezifikation: Ein CGM kann in drei verschiedenen Kodierungen vorliegen.

1. Binärkodierung (Binary Encoding), ein sehr kompaktes Format.
2. Klartextkodierung (Clear Text Encoding), lesbares, sehr umfangreiches Format, in dem auch editiert werden könnte.
3. Zeichenkodierung (Character Encoding), ein sehr kompaktes Format, das u.U. noch kompakter als die Binärkodierung ist. Bei der Zeichenkodierung werden nur die darstellbaren ASCII-Zeichen (94 oder 96 Zeichen in 7 Bit) verwendet, was ein Versenden über Netze vereinfacht.

Am häufigsten wird die Binärkodierung verwendet, die Zeichenkodierung ist dagegen kaum verbreitet.

CGM-Erzeuger sind folgende Programme, bzw. Grafikbibliotheken:

- FUGKS erzeugt zeichenkodierte Metafiles.
- XGKS erzeugt binärkodierte Metafiles.
- SunPHIGS kann binär- oder klartextkodierte CGM's erzeugen.
- Figaro+ mit PSO kann binär- oder klartextkodierte CGM's erzeugen.
- GRAZIL3D benutzt als Basisgrafikbibliothek SunPHIGS und kann somit entsprechende CGM's erzeugen.

Konvertierungstools:

Format		Lesen	Schreiben
cgm[a]	Computer Graphics Metafile	gplot	gplot
ps	(Colour) PostScript	-	gplot

[a] Konvertieren von binärkodiertem CGM in ein klartextkodiertes CGM und umgekehrt.

gplot

Gplot ist ein Programm zur Wandlung von Computer Graphics Metafiles, kurz CGM's, in andere Formate. Hauptanwendung ist die Konvertierung von CGM in PostScript, um ein CGM ausdrucken zu können. Gplot kann binär- oder klartextkodierte CGM's verarbeiten. Zum Previewen eines CGM's in einer X-Umgebung existiert eine Point-and-Click-Version von gplot, *gplotaw*, die auf den Sun-Rechnern zur Verfügung steht. Für Silicon Graphics ist eine Motif-Version, *gplotm*, vorhanden.

HPGL - Hewlett-Packard Graphics Language

Allgemein verwendete Endungen: .hp, .hpgl

Bemerkungen: HPGL ist ein Format, in dem hauptsächlich für Stiftplotter bestimmte Ausgaben gespeichert werden.

Probleme: HPGL kann keine Rasterbilder darstellen und die Möglichkeiten für Flächendarstellungen sind stark eingeschränkt.

HPGL-Erzeuger sind folgende Programme, bzw. Grafikbibliotheken:

- FUGKS
- Figaro+ mit PSO
- GRAZIL benutzt als Basisgrafikbibliothek FUGKS und kann somit Metafiles erzeugen.
- SAS-GRAPH (z.Z. nur an einer Stelle im ZIB verfügbar).

Mögliche Ausgabegeräte: Der *Diabelichter* und der *Thermotransferdrucker PhaserIII* können HPGL-Dateien interpretieren und darstellen.

Literatur zu GKS [6], CGM: s. [29].

3.1.3 Seitenbeschreibungssprachen

Eine *Seitenbeschreibungssprache* besteht aus einer Menge von Anweisungen, die rechner- und ausgabegeräteunabhängig das Aussehen einer (Druck-)Seite beschreiben. Eine Seite setzt sich zusammen aus:

- Text in verschieden Größen und Fonts,
- Grafikelementen wie Linien, Punkte und Flächen,
- kompletten Bildern (Rasterbildern).

Daraus sollte eine hochqualitative, evtl. auch farbige Ausgabe entstehen. Die Qualität ist nur durch die Güte des Ausgabegerätes beschränkt.

PostScript

Als de-facto-Standard der Seitenbeschreibungssprachen hat sich *PostScript*, eine Entwicklung von Adobe Systems Inc., durchgesetzt. Nahezu alle Grafik- und Textverarbeitungsprogramme (DTP-Programme) können PostScript erzeugen, und für sehr viele Ausgabegeräte (z.B. SPARCprinter) gibt es PostScript-Interpreter. Viele Ausgabegeräte verfügen über einen eigenen Prozessor und Hauptspeicher und interpretieren PostScript selbst (z.B. Apple-LaserWriter). PostScript-Ausgabegeräte sind *immer* Rastergeräte. Die Auflösung eines Rastergeräts wird in Punkten pro Zoll (Dots per Inch, dpi) angegeben. Bildschirme

haben eine Auflösung von 50 bis 100 dpi (90 dpi bei 15"-Sun-Bildschirmen). Typische Auflösungen von Druckern im Bürobereich sind 200 bis 600 dpi, im Bereich der kommerziellen Drucktechnik (Fotosatz) sind Auflösungen von 1200 dpi und mehr üblich.

PostScript ist eine komplette Programmiersprache mit Kontrollstrukturen, so daß erst bei der Interpretation das endgültige Aussehen der Ausgabe bestimmt wird. Die PostScript-Befehle werden *Operatoren* und ihre Parameter *Operanden* genannt. Die Versorgung der Operatoren mit Operanden erfolgt über einen Stack, auf den die Operanden abgelegt und für die nachfolgenden Operatoren nach dem LIFO-Prinzip herausgeholt und abgearbeitet werden.[2] Liefern Operatoren Ergebnisse zurück, so werden diese wieder auf den Stack gelegt. Ein Beispiel:

```
1 2 add ⇒ 3
```

Hierbei werden 1 und 2 addiert und das Ergebnis wird wieder auf dem Stack abgelegt. Kommentare sind in PostScript erlaubt. Sie beginnen mit einem % und gelten bis zum Ende der Zeile.

Generell sind im PostScript-Standard drei Implementierungsstufen vorgesehen:

1. *Level 1 PostScript* enthält die auf allen Geräten verfügbaren Operatoren, wie sie in [2] beschrieben sind.

2. *Level 2 PostScript* stellt ein erweitertes PostScript dar. Die wichtigsten Erweiterungen betreffen die Operatoren zur Farbbehandlung und zur Möglichkeit, Farbrasterbilder (`colorimage`) zu integrieren. Level 2 ist aufwärtskompatibel zu Level 1.

3. *Display PostScript* (DPS) ist ein um Operatoren zur Bildschirmsteuerung erweitertes PostScript, dessen Basis Level 1 oder 2 sein kann.

Im ZIB verarbeiten bis auf den zentralen Farbdrucker (PhaserIII) alle PostScript-fähigen Drucker, auch die SPARCprinter, Level 1 PostScript.

PostScript verwendet ein kartesisches Koordinatensystem, bei dem der Ursprung auf der linken unteren Ecke liegt. Bei einer A4-Seite wird immer vom Portrait-Format (hochkant) ausgegangen. Soll also A4-quer (Landscape-Format) ausgegeben werden, so muß mit PostScript-Operatoren die Seite gedreht werden. Das Papier quer einzulegen nützt nichts (siehe Kap. 4.2.2, Seite 83). Koordinaten werden in *Punkten* angegeben. Ein Punkt in PostScript entspricht $\frac{1}{72}$ Zoll ≈ 0.353 mm.

Obwohl PostScript neben der eigentlichen Syntax keine weiteren Strukturierungen vorschreibt, gibt es einige Empfehlungen und Konventionen, die beachtet werden sollten. So sollte die erste Zeile einer Datei, die PostScript-Anweisungen enthält, immer mit `%!` anfangen. Im Unix ist das eine sogenannte Magic-Number, die besagt, daß diese Datei eine PostScript-Datei[3] ist. Viele Druckkommandos (z.B. `lpr`) beinhalten ein Filterprogramm, das ASCII-

[2] Postfix-Notation

[3] eigentlich PostScript-Programm

Dateien in PostScript wandelt, so daß sie auf einem PostScript-Drucker ausgegeben werden können. Wenn es sich bei der zu druckenden Datei schon um eine PostScript-Datei handelt, darf der Filter sie nicht mehr wandeln. Beginnt die Datei mit diesen beiden Zeichen, wird der Filter nicht aktiv.

Darüberhinaus gibt es noch weitere Strukturierungsempfehlungen, die sogenannten *Document Structuring Conventions* (DSC). Darin wird der Aufbau einer PostScript-Datei beschrieben, wobei die einzelnen Teile durch genau definierte Kommentare getrennt und Hinweise über Seitenaufbau und Weiterverarbeitung gegeben werden. Wenn ein bestimmtes Mindestmaß dieser Konventionen eingehalten wird, dann bezeichnet man die PostScript-Datei als DSC-konform.

Eine PostScript-Datei, die in ein anderes Dokument integriert werden soll, muß einem bestimmten Format, dem *Encapsulated PostScript File Format* (EPSF) genügen, damit sie, wie gewünscht, im Dokument erscheint. Eine Encapsulated PostScript-Datei beschreibt nur *eine* Seite, muß bestimmte Strukturierungsanweisungen (DSC) enthalten und darf einige PostScript-Operatoren nicht enthalten. Die im Kapitel 2.4 beschriebenen Programme verarbeiten Encapsulated PostScript-Dateien, die mindestens folgenden Konventionen genügen müssen:

- Die Datei beginnt mit folgender Zeile:

  `%!PS-Adobe-3.0`

- Daran schließt sich folgende Zeile an:

  `%%BoundingBox:` *llx lly urx ury*

  llx lly = linke untere X- bzw. Y-Koordinate

  urx ury = rechte obere X- bzw. Y-Koordinate

  Hier sind die Koordinaten des Begrenzungsvierecks um die erzeugte „Seite“ (Bild) in Punkten (s.o.) anzugeben (herauszufinden durch Einlesen der PostScript-Datei in das Programm *ghostview* und Positionieren des Cursors auf den entsprechenden Eckpunkten).

- Einige Operatoren dürfen nicht verwendet werden:
  - `initgraphics, initmatrix, initclip`
  - `copypage, erasepage`
  - `grestoreall`
  - `showpage` (falls das importierende Programm diesen Operator nicht umdefiniert, was sehr oft der Fall ist).

Detaillierte Ausführungen zum Thema PostScript finden sich in [1, 3].

3.2 Raster Toolkits zur Bildbearbeitung

Zur Bearbeitung speziell von Rasterbildern gibt es eine ganze Reihe, zum Teil recht umfangreiche, Programmpakete, die sogenannten *Raster Toolkits*. Ein we-

sentlicher Zweck dieser Toolkits, nämlich die Konvertierung verschiedener Formate untereinander, wurde im letzten Abschnitt schon ausführlich behandelt. Hier nun sollen die weiteren Möglichkeiten angesprochen werden, die die Raster Toolkits bieten. Zunächst soll ein Überblick über das gesamte Spektrum der Funktionen gegeben werden. Danach werden die Toolkits einzeln vorgestellt.

Bei der folgenden Tabelle ist zu beachten, daß nicht jedes Toolkit alle hier aufgeführten Funktionalitäten bietet. Bei der Beschreibung der einzelnen Toolkits werden die jeweiligen Programme thematisch geordnet und einer der nachfolgend beschriebenen Kategorien zugeordnet. Die Zuordnung kann dabei natürlich nicht immer ganz eindeutig sein. Wir haben folgende Kategorien gewählt:

Formatkonvertierung
Auf das Thema Formatkonvertierung wurde im vorigem Abschnitt ausführlich eingegangen. Es gibt auch zum Teil generische Konvertierungstools, die das Eingabeformat und das gewünschte Ausgabeformat selbstständig erkennen. Oft wird in einem Raster Toolkit auch ein eigenes Format benutzt. Dann gibt es meistens eine Reihe von Programmen, die das eigene Format in andere umsetzen und umgekehrt.

Bilddarstellung
Einige Toolkits bieten die Möglichkeit, Rasterbilder auf unterschiedlicher Hardware, insbesondere auf dem Bildschirm, auszugeben.

Erstellen von Bildschirmkopien
Mit geeigneten Programmen lassen sich der ganze Bildschirm, Teile davon oder einzelne Windows in einer Bilddatei speichern.

Bildtransformation
Unter Bildtransformationen wollen wir Operationen wie Verkleinern, Vergrößern, Strecken, Stauchen, Scheren oder Drehen verstehen. Insbesondere das Skalieren von Rasterbildern ist in fast jedem Toolkit möglich. Oft läßt sich auch ein bestimmter Ausschnitt des Bildes herausgreifen (*cutting*).

Bildkomposition
Fast alle Toolkits bieten ebenfalls Möglichkeiten zur Bildkomposition. In der Regel können Bilder zumindest überlagert oder unter Benutzung des *alpha channels*[4] überblendet werden. Mit einigen Toolkits kann eine ganze Serie von Bildern verkleinert und zu einer Collage zusammengestellt werden. Die Komposition eines RGB-Bildes aus drei Graustufenbildern läßt sich ebenfalls hier einordnen.

Bildanalyse
Digitalisierte Bilder können auf vielfältige Art und Weise charakterisiert und analysiert werden. Zu erwähnen sind insbesondere Farbhistogramme, Farbmit-

[4] s. Alpha-Kanal im Glossar, Anhang B.

telwerte, Varianzen und Korrelationen.

Farbtransformation
Um ein Rasterbild auf einer bestimmten Hardware darzustellen, kann es notwendig sein, die Anzahl der Farben zu reduzieren. Zur Farbreduktion gibt es eine ganze Reihe teils recht ausgeklügelter Algorithmen. Neben der einfachen Farbquantisierung gibt es verschiedene *Halftoning* und *Dithering* Techniken. Umgekehrt kann man auch ein Graustufenbild mit Hilfe einer Farbtabelle zu einem Farbbild machen.

Pixelarithmetik
Es gibt eine Menge von Tools, die die Farbwerte der Eingabebilder addieren, subtrahieren, multiplizieren, interpolieren usw. Die Farbwerte können ebenfalls durch binäre Operationen *and*, *or* und *xor* verknüpft werden. All diese einfachen Operationen auf Pixelbasis lassen sich unter dem Sammelbegriff Pixelarithmetik zusammenfassen.

Bildverarbeitung
Unter dem Stichwort Bildverarbeitung lassen sich sehr viele Operationen anführen. Oft verwendet werden Kontrast- und Helligkeitskorrekturen oder eine Modifizierung der Farbbalance. Viele Toolkits bieten auch Operationen im Ortsbereich, zum Beispiel den Sobel-Filter oder den Laplace-Filter. Mit diesen Differenzenoperatoren kann zum Beispiel eine Kantenerkennung durchgeführt werden. Weniger verbreitet sind Operationen im Frequenzbereich, die eine Fourierzerlegung des Rasterbildes voraussetzen.

Bilderzeugung
Für bestimmte Operationen kann es sinnvoll sein, genau definierte Referenzbilder zu verwenden. Um zum Beispiel den Rand eines Bildes kontinuierlich von der Mitte her abzudunkeln, muß man eine Farbsubtraktion mit einem Referenzbild durchführen, das eine konzentrische Grauverteilung enthält. Neben solchen Referenzbildern lassen sich auch bestimmte Testbilder erzeugen.

Verschiedenes
Unter dieser Rubrik werden Programme aufgeführt, die sich schlecht in die obigen Gruppen einordnen lassen.

3.2.1 Image Magick

Das *Image Magick* Toolkit besteht aus nur sieben Programmen und ist somit sehr übersichtlich. Die einzelnen Tools sind recht leistungsfähig und können durch eine Vielzahl von Kommandozeilenoptionen gesteuert werden. Alle Tools können sämtliche unterstützen Rasterformate lesen bzw. schreiben. Ein eigenes Format gibt es nicht. Es werden ca. 15 verschiedene Formate unterstützt, die

auch fast alle direkt untereinander konvertiert werden können.[5] Einige wichtige, im Workstationbereich gebräuchliche Formate fehlen allerdings, so zum Beispiel das RGB-Format von Silicon Graphics oder das Wavefront RLA-Format. Hervorzuheben ist insbesondere das *animate*-Kommando, mit dem eine Sequenz von Bildern auf dem Bildschirm als Film dargestellt werden kann.

Formatkonvertierung

convert	-	convert an input file using one image format to an output file with a differing image format

Bilddarstellung

display	-	display an image on any workstation running X
animate	-	display a sequence of images on any workstation running X

Erstellen von Bildschirmkopien

XtoPS	-	capture some or all of an X server screen and save as EPS
import	-	capture some or all of an X server screen and save to file in MIFF image format

Bildtransformation

mogrify	-	transform an image or sequence of images

Bildkomposition

montage	-	create a composite image by combining several separate images

3.2.2 Iris Raster Tools

Auf allen Maschinen von Silicon Graphics gibt es einen umfangreichen Satz von Programmen zur Verarbeitung von Rasterbildern im RGB-Format von SGI. Diese Tools sind im Quellcode vorhanden, so daß sie als Vorlage für eigene Programme verwendet werden können. Das Toolkit macht insgesamt einen sehr heterogenen Eindruck. Oft gibt es mehrere Tools für ein und dieselbe Operation. Zu bemerken ist weiterhin, daß die Programme nicht die Umlenkung von Ein- und Ausgabe erlauben und somit nicht pipelinefähig sind. Trotzdem bietet das Toolkit einige sehr nützliche Möglichkeiten.

Formatkonvertierung

fromalias	-	convert an Alias image to an Iris image
frombin	-	create an RGB Iris image file from a binary dump of image data
fromcmap	-	convert a color map into an image with one scanline
fromcube	-	convert a Cubicomp/Vertigo image file to IRIS format
fromdi	-	convert an old .di dithered image into an RGB image
fromface	-	convert a UNIX faceserver image into IRIS format
fromgif	-	convert a GIF image into an IRIS image
frommac	-	convert a MacPaint image into an IRIS image
frompic	-	convert a MOVIE BYU .PIC image to an IRIS image
fromppm	-	convert an image in Jef Poskanzer's format into an IRIS image

[5] Wie erst nach Redaktionsschluß bekannt wurde, unterstützt die neueste Version von Image Magick noch weitere Formate. Insbesondere können in Verbindung mit `ghostscript` Postscript-Dokumente in verschiedene Rasterformate umgewandelt werden.

fromrla	-	convert a Wavefront image to an IRIS image
fromsun	-	convert a sun image into an IRIS image
fromtarga	-	convert a targa image into an IRIS image
fromxbm	-	convert an X Bitmap image into an IRIS image
fromyuv	-	convert an Abekas yuv image into an IRIS image
mapimg	-	translate a screen image into an RGB image
rle	-	force an image to be stored using run length encoding
toalias	-	convert an IRIS image to an Alias image
toascii	-	use text characters to represent an image
tobin	-	convert an Iris image to binary dump of pixel data
tobw	-	convert a color image to black and white
togif	-	convert an IRIS image to a Compuserve GIF image
tomac	-	convert an IRIS image to MacPaint format
tonews	-	convert an IRIS image into NeWS format
topict	-	convert an IRIS image to Macintosh PICT format
toppm	-	convert an IRIS image into Jef Poskanzer's ppm image format
tops	-	convert an iris image to PostScript
toscitex	-	convert IRIS images into Scitex CT2T images
tosun	-	convert an IRIS image to a sun raster file
totarga	-	convert from an IRIS image to a type 2 (RGB) targa image
toyuv	-	convert an IRIS image to yuv format
verbatim	-	force an image to be stored without run length encoding

Bilddarstellung

bgpaste	-	paste an image onto the root window
btree	-	display an image using a binary tree ordering
ipaste	-	display an image
movie	-	show a series of images in a sequence
rectimg	-	display a color or BW image on the iris
scope	-	explore an image of any size
slide	-	zoom an image up for full screen display

Erstellen von Bildschirmkopien

scrsave	-	save a part of the screen in an image file
snapshot	-	save a portion of the screen in an image file

Bildtransformation

fitimg	-	force an image to be a specific size
iflip	-	flip an image
invert	-	invert an image
iroll	-	roll an image in x and y directions
postcard	-	make an image look like a postcard
shear	-	shear an image diagonally
xzoom	-	magnify or minify an image in the x direction

Bildkomposition

addborder	-	surround an input image with a border image
addframe	-	add a border to an image
assemble	-	assemble an array of smaller images
cglue	-	create an rgb image out of 3 black and white images
fieldmerge	-	merge two field images into one frame
over	-	put one image on top of another
tile	-	repeats an image in two dimensions

Bildanalyse

abs	-	get the absolute value of an image
imean	-	find the average pixel value of an image

imgsize	-	print the size of an image
istat	-	print the header information of a list of image files
hist	-	compute and display the histogram of an image file
perhist	-	print percent histogram values for an image
vhist	-	display a 3-D volume histogram of a color image

Farbtransformation

duotone	-	make a color duotone image from a black and white image
gendit	-	perform general image dithering
quant	-	quantify an image
halftone	-	half-tone an image
histeq	-	histogram equalize an image file

Pixelarithmetik

add	-	add two images together
blend	-	linearly interpolates two images
iavg	-	average a set of images
imgwrap	-	shift pixels left one bit
max	-	get the maximum of two images
min	-	calculate the minimum of two images
mult	-	multiply two images
sub	-	subtract two images

Bildverarbeitung

addnoise	-	add noise to an image
blur	-	low pass filter an image
convolve	-	convolve an input image with a kernel
cscale	-	scale the rgb colors of an image
gammawarp	-	lighten or darken an image
hipass3	-	high pass filter an image
iblend	-	blend two images using a mat
noblack	-	remove all the black from an image
saturate	-	change an image's saturation
setlum	-	modifies the luminance on an image
subimg	-	extract a sub-region from an image
thresh	-	threshold one image with another

Bilderzeugung

conimg	-	create a constant image
dotgen	-	make an image of two crossed sinusoidal wave patterns
greyscale	-	make different patterns
randimg	-	generate a noise image

Verschiedenes

iset	-	set the type of an image
loadmap	-	loads the colormap from a file
oneband	-	get a single band of an image
repcolor	-	replace specified colors within an image
savemap	-	saves the current contents of the colormap

3.2.3 FBM

Die *FBM* oder *Fuzzy PixMap* Library enthält ungefähr zwanzig Programme zur Manipulation von Rasterbildern. Bei diesem Toolkit können wiederum alle Programme jedes unterstützte Format lesen bzw. schreiben. Die Liste der unterstützten Formate ist allerdings mit sechs oder sieben Formaten recht kurz.

Empfehlung: FBM zusammen mit dem *PBMPLUS* Toolkit benutzen.

Formatkonvertierung

fbcat	-	copy image (used for format conversion)
fbps	-	convert to PostScript
pbm2ps	-	convert PBM file to PostScript
raw2fbm	-	convert raw file to FBM format (eg. Amiga Digiview files)

Bildtransformation

fbext	-	extract region, resize, change aspect ratio
fbrot	-	rotate 90, 180, or 270 degrees

Bildkomposition

fbmask	-	set region to gray value

Bildanalyse

fbhist	-	compute histogram
fbinfo	-	dump image header

Farbtransformation

clr2gray	-	convert color to grayscale
fbhalf	-	halftone grayscale image (blue noise, Floyd-Steinberg, etc)
fbquant	-	color quantization (24 bit to 8..256 colors) Mod. Heckbert
gray2clr	-	add a gray colormap to a grayscale image

Bildverarbeitung

fbclean	-	flip isolated pixels (clean image)
fbedge	-	compute derivative image (edge detection)
fbnorm	-	normalize image intensity / increase contrast
fbsharp	-	sharpen by digital Laplacian (edge enhancement)

Verschiedenes

idiff	-	(and udiff) convert raw byte stream into byte difference
pbmtitle	-	add a title to a PBM file

3.2.4 PBMPLUS

PBMPLUS ist ein sehr umfangreiches Toolkit. Es werden drei eigene Formate benutzt, je nach verwendeter Farbtiefe *pbm* (portable bitmap), *pgm* (portable graymap) oder *ppm* (portable pixmap). Namen von Tools, die nur mit einem dieser Formate arbeiten, beginnen mit der entsprechenden Abkürzung; Namen von Tools, die alle drei Formate verarbeiten können, beginnen mit der Abkürzung *pnm* (portable anymap).

Formatkonvertierung

anytopnm	-	attempt to convert an unknown type of image file to a portable anymap
brushtopbm	-	convert a doodle brush file into a portable bitmap
cmuwmtopbm	-	convert a CMU window manager bitmap into a portable bitmap
fitstopgm	-	convert a FITS file into a portable graymap
fstopgm	-	convert a Usenix FaceSaver file into a portable graymap
g3topbm	-	convert a Group 3 FAX file into a portable bitmap
gemtopbm	-	convert a GEM .img file into a portable bitmap

giftoppm - convert a GIF file into a portable pixmap
gouldtoppm - convert Gould scanner file into a portable pixmap
hipstopgm - convert a HIPS file into a portable graymap
icontopbm - convert a Sun icon into a portable bitmap
ilbmtoppm - convert IFF ILBM file into a portable pixmap
imgtoppm - convert an Img-whatnot file into a portable pixmap
lispmtopgm - convert a Lisp Machine bitmap file into pgm format
macptopbm - convert a MacPaint file into a portable bitmap
mgrtopbm - convert a MGR bitmap into a portable bitmap
mtvtoppm - convert output from the MTV or PRT ray tracers into a portable pixmap
pbmto10x - convert a portable bitmap into Gemini 10X printer graphics
pbmtoascii - convert a portable bitmap into ASCII graphics
pbmtobg - convert a portable bitmap into BitGraph graphics
pbmtocmuwm - convert a portable bitmap into a CMU window manager bitmap
pbmtoepson - convert a portable bitmap into Epson printer graphics
pbmtog3 - convert a portable bitmap into a Group 3 FAX file
pbmtogem - convert a portable bitmap into a GEM .img file
pbmtogo - convert a portable bitmap into compressed GraphOn graphics
pbmtoicon - convert a portable bitmap into a Sun icon
pbmtolj - convert a portable bitmap into HP LaserJet format
pbmtomacp - convert a portable bitmap into a MacPaint file
pbmtomgr - convert a portable bitmap into a MGR bitmap
pbmtopi3 - convert a portable bitmap into an Atari Degas .pi3 file
pbmtoptx - convert a portable bitmap into Printronix printer graphics
pbmtox10bm - convert a portable bitmap into an X10 bitmap
pbmtoxbm - convert a portable bitmap into an X11 bitmap
pbmtozinc - convert a portable bitmap into a Zinc bitmap
pbmupc - create a Universal Product Code bitmap
pcxtoppm - convert a PCX file into a portable pixmap
pgmtofits - convert a portable graymap into FITS format
pgmtofs - convert portable graymap to Usenix FaceSaver format
pgmtolispm - convert a portable graymap into Lisp Machine format
pgmtopbm - convert a portable graymap into a portable bitmap
pgmtops - convert portable graymap to Encapsulated PostScript
pgmtoybm - convert a portable bitmap into a Bennet Yee „face“ file
pi1toppm - convert an Atari Degas .pi1 into a portable pixmap
pi3topbm - convert an Atari Degas .pi3 file into a portable bitmap
picttoppm - convert a MacIntosh PICT file into a portable pixmap
pnmnoraw - force a portable anymap into plain format
pnmtorast - convert a portable pixmap into a Sun raster file
pnmtotiff - convert a a portable anymap into a TIFF file
pnmtoxwd - convert a portable anymap into an X11 window dump
ppmtogif - convert a portable pixmap into a GIF file
ppmtoicr - convert a portable pixmap into NCSA ICR format
ppmtoilbm - convert a portable pixmap into an IFF ILBM file
ppmtopcx - convert a portable pixmap into a PCX file
ppmtopgm - convert a portable pixmap into a portable graymap
ppmtopi1 - convert a portable pixmap into an Atari Degas .pi1 file
ppmtopict - convert a portable pixmap into a Macintosh PICT file
ppmtops - convert a portable pixmap into color Encapsulated PostScript
ppmtopuzz - convert a portable pixmap into an X11 „puzzle“ file
ppmtotga - convert portable pixmap into a TrueVision Targa file
ppmtouil - convert a portable pixmap into a Motif UIL icon file
ppmtoxpm - convert a portable pixmap into an X11 pixmap
psidtopgm - convert PostScript „image“ data into a portable graymap
qrttoppm - convert output from the QRT ray tracer into a portable pixmap

rasttopnm	-	convert a Sun raster file into a portable anymap
rawtopgm	-	convert raw grayscale bytes into a portable graymap
rawtoppm	-	convert raw RGB bytes into a portable pixmap
spctoppm	-	convert an Atari compressed Spectrum file into a portable pixmap
sputoppm	-	convert an Atari uncompressed Spectrum file into a portable pixmap
tgatoppm	-	convert TrueVision Targa file into a portable pixmap
tifftopnm	-	convert a TIFF file into a portable anymap
xbmtopbm	-	convert an X11 or X10 bitmap into a portable bitmap
ximtoppm	-	convert an Xim file into a portable pixmap
xpmtoppm	-	convert an X11 pixmap into a portable pixmap
xwdtopnm	-	convert a X11 or X10 window dump file into a portable anymap
ybmtopbm	-	convert a Bennet Yee „face“ file into a portable bitmap

Bildtransformation

pbmreduce	-	read a portable bitmap and reduce it N times
pnmcrop	-	crop a portable anymap
pnmcut	-	cut a rectangle out of a portable anymap
pnmenlarge	-	read a portable anymap and enlarge it N times
pnmflip	-	perform one or more flip operations on a portable anymap
pnminvert	-	invert a portable anymap
pnmmargin	-	add a margin to a portable anymap
pnmrotate	-	rotate a portable anymap by some angle
pnmscale	-	scale a portable anymap
pnmshear	-	shear a portable anymap by some angle

Bildkomposition

pbmpaste	-	paste a rectangle into a portable bitmap
pnmcat	-	concatenate portable anymaps
pnmindex	-	build a visual index of a bunch of anymaps
pnmpaste	-	paste a rectangle into a portable anymap
pnmtile	-	replicate a portable anymap into a specified size

Bildanalyse

pgmhist	-	print a histogram of the values in a portable graymap
pgmtxtur	-	calculate textural features on a portable graymap
pnmfile	-	describe a portable anymap
ppmhist	-	print a histogram of a portable pixmap

Farbtransformation

pgmtoppm	-	colorize a portable graymap into a portable pixmap
ppmquant	-	quantize the colors in a portable pixmap down to a specified number
ppmquantall	-	run ppmquant on a bunch of files all at once, so they share a common colormap
ppmtorgb3	-	separate a portable pixmap into three portable graymaps
rgb3toppm	-	combine three portable graymaps into one portable pixmap

Pixelarithmetik

pnmarith	-	perform arithmetic on two portable anymaps
pnmconvol	-	general MxN convolution on a portable anymap

Bildverarbeitung

pgmbentley	-	Bentleyize a portable graymap
pgmedge	-	edge-detect a portable graymap
pgmenhance	-	edge-enhance a portable graymap
pgmnorm	-	normalize the contrast in a portable graymap

pgmoil	-	turn a portable graymap into an oil painting
pnmdepth	-	change the maxval in a portable anymap
pnmgamma	-	perform gamma correction on a portable anymap
pnmsmooth	-	smooth out an image
ppmrelief	-	run a Laplacian Relief filter on a portable pixmap

Bilderzeugung

pbmmake	-	create a blank bitmap of a specified size
pbmmask	-	create a mask bitmap from a regular bitmap
pgmramp	-	generate a grayscale ramp
ppmmake	-	create a pixmap of a specified size

Verschiedenes

pbmlife	-	apply Conway's rules of Life to a portable bitmap
pbmtext	-	render text into a bitmap
ppmpat	-	make a pretty pixmap

3.2.5 SDSC Image Tools

Die *SDSC Image Tools* waren ursprünglich nur zur Formatkonvertierung gedacht. Sie enthalten ein sehr leistungsfähiges generisches Konvertierungstool *imconv*, das direkt zwischen 20 verschiedenen Formaten wandeln kann. Das Eingabeformat sowie das gewünschte Ausgabeformat werden selbstständig erkannt, falls man nicht Optionen explizite angibt. Seit Version 2.0 kann man darüberhinaus Bilder überlagern, skalieren und Ausschnitte aus Bildern herausschneiden. Das Toolkit stammt vom *San Diego Supercomputer Center* (SDSC). Zur Zeit werden nur Binärversionen weitergegeben, nicht der Quellcode.

Formatkonvertierung

imconv	-	convert between image file formats

Bildtransformation

imcopy	-	copy a portion of an image to a new file
imflip	-	flip images vertically or horizontally and store in a new file
imshear	-	shear an image horizontally or vertically
imroll	-	cycle an image horizontally or vertically
imrotate	-	free rotate an image
imscale	-	scale an image up or down and save it in a new file

Bildkomposition

imcomp	-	digitally composite images
imfill	-	fill a region of an image with a color or gradient
impaste	-	paste an image atop a background and store in a new file

Bildanalyse

imfile	-	discern the image format of a file(s)
imhist	-	compute an image histogram

Farbtransformation

imcltroll	-	cycle a color lookup table
imgray	-	convert an image to grayscale
immono	-	convert an image to monochrome

Verschiedenes

imcat	-	concatenate images into multi-image files
imformats	-	list information on image file formats

3.2.6 Utah Raster Toolkit

Dieses Toolkit ist wiederum recht umfangreich. Die einzelnen Programme arbeiten alle mit einem eigenen Format, dem *Utah rle-Format*. *Rle* steht für run-length encoding. Alle Programme sind pipelinefähig. Erwähnenswert ist, daß bei der Ausgabe Rasterfiles die auf „.Z“ enden, automatisch mit *compress* komprimiert werden. Analog können komprimierte Files auch direkt gelesen werden.

Formatkonvertierung

cubitorle	-	convert cubicomp image to an RLE format file
dvirle	-	convert dvi version 2 files, produced by TeX82, to RLE images
giftorle	-	convert GIF images to RLE format
iristorle	-	convert Silicon Graphics image files into an RLE file
painttorle	-	convert MacPaint images to RLE format
pgmtorle	-	convert a pbmplus/pgm image file into an RLE image file
ppmtorle	-	convert a PBMPLUS/ppm image file into an RLE image file
rastorle	-	convert sun raster file image to an RLE format file
rawtorle	-	convert raw image data to RLE
rlatorle	-	convert a Wavefront „rlb“ image file into an RLE image file
rletoascii	-	print an RLE image as ASCII chars
rletogif	-	convert RLE files to GIF format
rletoiris	-	convert RLE files to Silicon Graphics image file
rletopaint	-	convert an RLE file to MacPaint format using dithering
rletoppm	-	convert a Utah RLE image file into a PBM-PLUS/ppm image file
rletops	-	convert RLE images to PostScript
rletorast	-	convert an RLE file to a Sun rasterfile
rletoraw	-	convert RLE file to raw RGB form
rletorla	-	convert a Utah RLE image file into a Wavefront „rlb“ image file
rletotiff	-	convert 24 bit RLE image files to TIFF
targatorle	-	convert Targa 32 TIPS images to RLE format
tifftorle	-	convert 24 bit TIFF image files to RLE
tobw	-	convert a 24 bit RLE file to eight bits of gray scale value
unexp	-	convert „exponential“ files into normal files
wasatchrle	-	convert Wasatch Systems image files to RLE format

Bilddarstellung

get_orion	-	get RLE images to an Orion graphics display
getap	-	get RLE images to an Apollo display
getbob	-	display RLE files on HP Bobcat screens
getfb	-	display an RLE file on a BRL libfb frame buffer
getgmr	-	restore an RLE image to a Grinnell GMR-27 frame buffer
getiris	-	display an RLE image on a Silicon Graphics Iris Workstation
getmac	-	display RLE images on a MacIntosh display
getmex	-	get RLE images to an Iris display under the window manager
getqcr	-	photograph an RLE image with the Matrix QCR-Z camera
getren	-	get RLE images to an HP98721 („Renaissance“) display
getsun	-	get RLE images to a sun window
gettaac	-	display an RLE image on a Sun TAAC-1

Bildtransformation

crop	-	change the size of an RLE image

fant	-	perform simple spatial transforms on an image
repos	-	reposition an RLE image
rleflip	-	invert, reflect or rotate an image
rlezoom	-	magnify an RLE file by pixel replication

Bildkomposition

graytorle	-	merges gray scale images into an RLE format file
into	-	copy into a file without destroying it
rlecat	-	concatenate and repeat images
rlecomp	-	digital image compositor
rlepatch	-	patch smaller RLE files over a larger image
rlesplice	-	splice two RLE files together horizontally or vertically
rlesplit	-	split a file of concatenated RLE images into separate image files
unslice	-	quickly assemble image slices

Bildanalyse

rlebox	-	print bounding box for image in an RLE file
rlehdr	-	prints the header of an RLE file
rlehisto	-	generate histogram of RLE image
rleprint	-	print the values of all the pixels in the file

Farbtransformation

applymap	-	apply the color map in an RLE file to the pixel data
mcut	-	quantize colors in an image using the median cut algorithm
mergechan	-	merge channels from several RLE files into a single output stream
rledither	-	Floyd Steinberg dither an image to the given colors
rleldmap	-	load a new color map into an RLE file
rlequant	-	variance based color quantization for RLE images
rlescale	-	produce gray scale images
rletogray	-	splits an RLE format file into gray scale images

Bildverarbeitung

pyrmask	-	blend two images together using Gaussian pyramids
rlenoise	-	add random noise to an image
rlespiff	-	use simple contrast enhancement to „spiff up“ an image
rleswap	-	swap the channels in an RLE file
smush	-	defocus an RLE image

Bilderzeugung

rleClock	-	generate a clock face in RLE format
rlebg	-	generate simple backgrounds
rlemandl	-	compute images of the Mandelbrot set

Verschiedenes

rleaddcom	-	add picture comments to an RLE file
rleaddeof	-	put an end of image marker on an RLE file
rleselect	-	select images from an RLE file
rlesetbg	-	set the background value in the RLE header
rlestereo	-	produce anaglyph from stereo pair

3.2.7 Beispiele zur Bildverarbeitung mit Raster Toolkits

Im folgenden soll anhand eines kleinen Beispiels gezeigt werden, wie verschiedene Raster Toolkits zur Bildverarbeitung eingesetzt werden können. Fast alle Aufgaben lassen sich besser und bequemer mit einem professionellen Paint-Programm bewerkstelligen. Für kleinere Aufgaben bietet sich das Arbeiten mit den Raster Toolkits aber dennoch an, da diese fast an jedem Rechner zur Verfügung stehen und einfach zu handhaben sind. Die Kommandos lassen sich problemlos in Shell-Scripten benutzen. Dies bringt insbesondere bei der Bearbeitung von längeren Bildsequenzen große Vorteile mit sich. Nützlich ist auch die Möglichkeit, mehrere Kommandos über den Pipe-Mechanismus verketten zu können.

Bei unserer Beschreibung beziehen wir uns auf die Bilder D.16 bis D.22 im Anhang. Ausgangspunkt für unser Beispiel sind die Bilder D.16 und D.19, die eine Teekanne und ein Martini-Glas jeweils auf einem einfarbigen Hintergrund darstellen. Mit Hilfe geeigneter Kommandos soll zunächst die Teekanne aus der Mitte des Bildes in die linke obere Ecke verschoben werden. Dann soll das Martini-Glas eingeblendet und schließlich noch ein Schriftzug überlagert werden.

Die Teekanne liegt als Rasterfile `teapot.ppm` im ppm-Format vor. Ein Bild in diesem Format kann mit dem Kommando `display` angezeigt werden:

```
% display teapot.ppm
```

Mit diesem Programm können Pixelpositionen relativ zur linken oberen Ecke ermittelt werden. Wir erhalten so die Koordinaten für denjenigen rechteckigen Bildausschnitt, der die Teekanne gerade noch vollständig enthält. Außerdem läßt sich mit `display` für jedes Pixel der Farbwert bestimmen.

Mit dem folgendem Kommando wird der Bildausschnitt, der die Teekanne enthält, kopiert und als Rasterfile `cut.rle` im rle-Format gespeichert:

```
% imcopy teapot.ppm cut.rle -xposition 220 -yposition 98
  -xsize 402 -ysize 232
```

Als nächstes wird ein leeres Bild gleicher Größe mit der zuvor bestimmten Hintergrundfarbe erzeugt:

```
% rlebg -s 402 232 49 52 88 > background.rle
```

Mit diesem Bild wird die Teekanne an ihrer alten Position abgedeckt. Das Ergebnis soll unter dem Namen `empty.rle` gespeichert werden:

```
% impaste background.rle teapot.ppm empty.rle -xposition 220
  -yposition 98
```

Nun kann der vorher kopierte Ausschnitt links oben wieder eingefügt werden:

```
% impaste cut.rle empty.rle shifted.rle -xposition 107
  -yposition 10
```

Das Martini-Glas liegt zunächst als rgb-File `martini.rgb` vor. Dieses Format wird von `display` nicht unterstützt. Stattdessen werden die Farbwerte und Pixelpositionen mit dem Kommando `scope` bestimmt:

```
% scope martini.rgb
```

Das `scope`-Kommando steht allerdings nur auf Silicon-Graphics-Maschinen zur Verfügung. Auf anderen Plattformen muß ein rgb-File zum Beispiel mit `imconv` in ein Format gewandelt werden, welches von `display` unterstützt wird.

Um das Martini-Glas im Vergleich zur Teekanne auf die richtige Größe zu bringen, führen wir zunächst eine Skalierung durch und speichern das Ergebnis unter dem Namen `martini.rle` im rle-Format:

```
% imscale martini.rgb -scale 0.45 martini.rle
```

Um später das Glas in ein anderes Bild einblenden zu können, muß zunächst einmal ein Alpha-Kanal generiert werden. Dieser gibt für jedes Pixel einen Transparenzwert an, der beim Überblenden berücksichtigt wird. Wir beginnen damit, das Martini-Glas in ein Bitmap-Format zu konvertieren. Jedes Pixel mit einem Grauwert[6] größer gleich eins soll auf weiß abgebildet werden:

```
% immono martini.rle -threshold 1 mask.xbm
```

Wie in Bild D.20 zu erkennen ist, sind im Inneren des Glases einige völlig schwarze Pixel mit einem Grauwert von null zurückgeblieben. Diese Stellen müssen nachträglich per Hand ausradiert werden. Dazu läßt sich beispielsweise der Icon-Editor `bitmap` verwenden:

```
% bitmap mask.xbm
```

Als nächstes muß die Bitmap invertiert werden, damit der Hintergrund einem maximalen Alpha-Wert bekommt und somit vollständig transparent wird. Gleichzeitig glätten wir den Übergang von hell nach dunkel etwas aus. Dadurch werden später Alias-Effekte reduziert.

```
% xbmtopbm maske.xbm | pnminvert | pnmsmooth | pgmtorle |
  rletoraw > alpha.raw
```

Der Alpha-Kanal liegt jetzt als unformatierte Folge von Bytes vor. Wir wandeln nun auch das Martini-Glas in solch eine Bytefolge um, wobei die Option `-N` dafür sorgt, daß die roten, grünen und blauen Teilbilder nacheinander ausgegeben werden:

```
% rletoraw -N martini.rle > martini.raw
```

Nun können alle Teilbilder zu einem vollständigen Rasterbild mit Alpha-Kanal zusammengefügt werden. Dabei muß die Bildgröße explizit angegeben werden:

```
% cat martini.raw alpha.raw | rawtorle -N -w 314 -h 361 -n 4
  > martini.rle
```

[6] Der Grauwert ist eine Zahl zwischen 0 und 255. Er wird in der Regel aus einem RGB-Farbtripel über die sogenannte NTSC-Formel $0.30R + 0.59G + 0.11B$ berechnet.

Das Überblenden ist am einfachsten, wenn Hintergrund- und Vordergrundbild die gleiche Größe haben. Deshalb erzeugen wir ein vollständig transparentes Hintergrundbild von passender Größe,

```
% rlebg -s 800 538 0 0 0 0 > background.rle
```

in das wir das Martini-Glas an der richtigen Stelle hineinsetzen:

```
% impaste martini.rle background.rle big.rle -xposition 400
  -yposition 150
```

Nun kann die eigentliche Überblendung durchgeführt werden:

```
% rlecomp big.rle over shifted.rle > paste.rle
```

Abschließend soll noch ein Schriftzug eingeblendet werden. Dazu muß der Text mit einem Textverarbeitungsprogramm in Form einer Postscript-Datei bereitgestellt werden. Dies kann beispielsweise mit TEX, Framemaker oder Showcase geschehen. Aus der Postscript-Datei `magic.ps` wird mit Hilfe eines geeigneten Interpreters ein Rasterbild `magic.rgb` erzeugt:

```
% gs -sDEVICE=rgb -g500x200 -r144x144 -dSmooth=2
  -sOutputFile=page.rgb -- magic.ps
```

Auf dem Rasterbild ist in der Regel eine ganze Papierseite dargestellt. Mit dem folgenden Kommando wird deshalb genau der gewünschte Text herausgeschnitten:

```
% imcopy page.rgb magic.rle -xposition 20 -yposition 50
  -xsize 410 -ysize 130
```

Der Text muß jetzt noch auf die richtige Größe gebracht werden:

```
% imscale magic.rle -xsize 300 -ysize 200 scale.rle
```

Das Erzeugen eines Alphakanals und die eigentliche Überblendung erfolgt ganz analog zu den oben diskutierten Arbeitsschritten:

```
% immono scale.rle -threshold 254 alpha.pbm
% pnmsmooth alpha.pbm | pgmtorle | rletoraw > alpha.raw
% rletoraw -N scale.rle > scale.raw
% cat scale.raw alpha.raw | rawtorle -N -w 300 -h 200 -n 4
  > magic.rle
% rlebg -s 800 538 0 0 0 0 > background.rle
% impaste magic.rle background.rle big.rle -xposition 80
  -yposition 300
% rlecomp big.rle over paste.rle > final.rle
```

Was im Zeitalter der technischen Reproduzierbarkeit des Kunstwerks verkümmert, das ist seine Aura.

– Walter Benjamin, *Das Kunstwerk im Zeitalter seiner technischen Reproduzierbarkeit*, Zeitschrift für Sozialforschung 1, 1936

Kapitel 4

Ein- und Ausgabe von Bildern

4.1 Bildschirm

4.1.1 Eingabe (Bildspeicherung)

Im folgenden werden Programme beschrieben, mit denen auf dem Bildschirm angezeigte Bilder in eine Datei gesichert werden können. Wenn möglich, sollte man aus dem Programm heraus, welches die Bilder erzeugt, direkt abspeichern. Nur wenn das nicht vorgesehen ist oder das gewünschte Dateiformat auf diese Weise nicht erzeugt werden kann, muß auf die hier beschriebenen Hilfsprogramme zurückgegriffen werden. Bei den dabei erzeugten Dateien handelt es sich ausschließlich um Rasterdateien.

screendump **Sun**

Screendump sichert den gesamten Bildschirminhalt oder einen durch Bildschirm-Koordinaten anzugebenden Ausschnitt in eine Datei. Direkte Druckerausgabe kann wie folgt erreicht werden:
screendump | lpr -v

Aufruf	`screendump [`*Optionen*`] [`*Dateiname*`]`
erzeugte(s) Dateiformat(e)	`Sun-Raster`

snapshot **Sun**

Snapshot ist ein interaktives Programm zum Sichern von Bildschirminhalten. Es kann der ganze Bildschirminhalt, ein einzelnes Fenster oder ein beliebiger Ausschnitt gesichert werden. Snapshot verfügt über einen Timer, so daß das Sichern nicht unmittelbar nach Angabe des zu sichernden Ausschnitts erfolgt. Mit snapshot können gesicherte Rasterdateien gedruckt oder angezeigt werden.

Besonderheiten	Nur auf Sun's unter OpenWindow
Aufruf	`snapshot [`*Optionen*`]`
erzeugte(s) Dateiformat(e)	`Sun-Raster`

snapshot — SGI

Snapshot ist ein interaktives Programm zum Sichern von Bildschirminhalten. Der zu sichernde Bildschirmausschnitt wird durch Aufziehen eines Rahmens angegeben. Mit snapshot können gesicherte Rasterdateien angezeigt werden.

Aufruf	`snapshot [-b]`
erzeugte(s) Dateiformat(e)	`SGI RGB-Image`

xgrab — Sun

Xgrab ist eine interaktive Benutzungsoberfläche auf der Basis von X11 für *xgrabsc (s.u.)*.

Aufruf	`xgrab`
erzeugte(s) Dateiformat(e)	`XWD, X Bitmap, X Pixmap, PostScript, Encapsulated PostScript`

xgrabsc — Sun

Xgrabsc kann den ganzen Bildschirminhalt, ein einzelnes Fenster oder einen beliebigen Ausschnitt sichern. Xgrabsc verfügt über einen Timer, so daß das Sichern nicht unmittelbar nach Angabe des zu sichernden Ausschnitts erfolgt. Zusätzlich zum Sichern kann das Rasterbild noch manipuliert (z.B. aufgehellt) und in verschiedene Formate gewandelt werden.

Aufruf	`xgrabsc [`*Optionen*`]`
erzeugte(s) Dateiformat(e)	`XWD, X Bitmap, X Pixmap, PostScript, Encapsulated PostScript`

imgsnap — SGI

Imgsnap ist ein interaktives Programm zum Sichern von Bildschirminhalten. Der zu sichernde Bildschirmausschnitt wird durch Aufziehen eines Rahmens angegeben. Mit imgsnap können gesicherte Rasterdateien angezeigt werden.

Aufruf	`imgsnap [`*Dateiname*`]`
erzeugte(s) Dateiformat(e)	`SGI RGB-Image, TIFF, FIT`

scrsave **SGI**

Scrsave sichert den gesamten Bildschirminhalt oder einen durch Bildschirm-Koordinaten anzugebenden Ausschnitt in eine Datei.

Aufruf	`scrsave` *Dateiname* `[`*X-, Y-Koordinaten*`] [-b]`
erzeugte(s) Dateiformat(e)	`SGI RGB-Image`

xwd **Sun, SGI**

Mit xwd wird ein X-Fenster gesichert. Die Auswahl des zu sichernden Fensters geschieht durch Anklicken nach Aufruf von xwd.

Aufruf	`xwd [`*Optionen*`]`
erzeugte(s) Dateiformat(e)	`XWD (X11 Window Dump Format)`

4.1.2 Ausgabe (Previewing)

Unter Previewing versteht man das Ansehen von Bildern auf dem Bildschirm, die in irgendeiner Datei in irgendeinem Bildformat vorliegen. Im ZIB sind eine ganze Reihe von Preview-Programme für die unterschiedlichsten Formate vorhanden. In den meisten Fällen können damit Rasterbilder angesehen werden. Mit einigen Previewern können die eingelesenen Bilder noch geändert werden. Die in einer Bilddatei vorhandene Farbtiefe kann natürlich nur dann ohne Einschränkungen dargestellt werden, wenn der Bildschirm diese auch unterstützt.

animate [Image Magick] **Sun**

Interaktive Darstellung einer Serie von Dateien im MIFF-Format. Bei Rasterbildern in unterschiedlicher Größe werden alle Bilder auf die Größe des kleinsten Bildes abgeschnitten. Diese Dateien können mit *compress* komprimiert sein. Es sollten nur soviele Rasterbilder angegeben werden, wie in den Hauptspeicher (ohne Swappen) passen.

Aufruf	`animate [`*Optionen*`] [`*Dateiname(n)*`]`
Dateiformate	`MIFF, *.Z`

display [Image Magick] **Sun**

Interaktive Darstellung von Dateien im MIFF-Format. Diese Dateien können mit *compress* komprimiert sein.

Aufruf	`display [`*Optionen*`] [`*Dateiname(n)*`]`
Dateiformate	`MIFF, *.Z`

getsun [Utah Raster Toolkit] **Sun**

Darstellung eines Rasterbildes in einem SunView-Fenster mit entsprechend eingeschränkter Farbdarstellung (max. 256 Farben).

Aufruf	`getsun [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`RLE`

gplotaw **Sun**

gplotaw ist eine Point-and-Click-Version von gplot zum Previewen von CGM's, jedoch ohne Konvertiermöglichkeiten in ein anderes Format. gplotaw ist eine Entwicklung des Pittsburgh Supercomputing Center.

Aufruf	`gplotaw`
Dateiformate	`Binär- oder klartextkodierte Computer Graphics Metafiles (CGM's)`

pageview **Sun**

Pageview ist ein interaktiver PostScript-Previewer, in dem die PostScript-Datei auch noch editiert werden kann.

Besonderheiten	Nur auf Sun's unter OpenWindow
Aufruf	`pageview [`*Optionen*`] [`*PostScript-Datei*`]`
Dateiformate	`PostScript`

screenload **Sun**

Darstellen eines Rasterbildes.

Aufruf	`screenload [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`Sun-Raster`

xli, xlito, xloadimage, xview, xsetbg **Sun**

Darstellung von Rasterbildern in vielen verschiedenen Rasterformaten. *Xloadimage*, *xli* und *xview* zeigen die Rasterbilder in einem Fenster. *Xsetbg* stellt das Bild auf dem Bildschirmhintergrund vergrößert oder mehrmals ("Tiling") bildschirmfüllend dar. Mit *xlito* können zu *xli* nachträglich Darstellungsoptionen gesetzt bzw. verändert werden. Wenn mehrere Bilder darzustellen sind, dann können sie automatisch in gewissen Zeitabständen in Serie dargestellt werden ("Slideshow"). Die Dateien können mit *compress* komprimiert sein.

Aufruf	`xloadimage [`*Optionen*`] [`*Dateiname(n)*`],`
	`xli [`*Optionen*`] [`*Dateiname(n)*`],`
	`xview [`*Optionen*`] [`*Dateiname(n)*`],`
	`xsetbg [`*Optionen*`] [`*Dateiname(n)*`],`
	`xlito [`*Optionen*`] [`*Dateiname(n)*`]`

Dateiformate	`FBM, GEM, GIF, G3 FAX, JFIF, MacPaint, PCX, PBM, PGM, PPM, Sun Raster, RLE, X Pixmap, X11 Bitmap, XWD, *.Z`

xtex **Sun**

Interaktive Darstellung einer .dvi-Datei.

Aufruf	`xtex [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`DVI`

get4d [Utah Raster Toolkit] **SGI**

Darstellung eines Rasterbildes. Ist das Bild größer als der Bildschirm, kann mit der Maus der Mittelpunkt des Bildausschnittes verschoben werden ("Panning").

Aufruf	`get4d [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`RLE`

gplotm **SGI**

gplotm ist eine Point and Click Version von gplot zum Previewen von CGM's, jedoch ohne Konvertiermöglichkeiten in ein anderes Format. gplotm ist eine Entwicklung des Pittsburgh Supercomputing Center.

Aufruf	`gplotm`
Dateiformate	`Binär- oder klartextkodierte Computer Graphics Metafiles (CGM's)`

imf_dspl [Wavefront Advanced Visualizer] **SGI**

Display-Programm des Wavefront Advanced Visualizer's. Es können auch mehrere Rasterbilder dargestellt werden, wenn die Dateinamen vor dem Suffix eine vierstellige Nummer haben und bei den Optionen die Nummer des ersten und des letzten darzustellenden Bildes und der Dateiname *ohne* Nummer angegeben wurden.

Besonderheiten	Es besteht nur eine Lizenz auf grafs21
Aufruf	`imf_dspl [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`Wavefront RLA, TIFF, TARGA, SGI RGB-Image, ...`
Dokumentation	`Handbuch Advanced Visualizer`

imgview **SGI**

Interaktive Darstellung von einem oder mehreren Rasterbildern. Mit der Maus kann über größere Bilder navigiert werden.

Aufruf	`imgview [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`SGI RGB-Image, TIFF, FIT`

ipaste **SGI**

Darstellen eines Rasterbildes.

Aufruf	`ipaste [`*Optionen*`]` *Dateiname*
Dateiformate	`SGI RGB-Image`

movie **SGI**

Movie stellt eine Serie von Rasterbildern als Film dar. Alle Rasterbilder müssen die gleiche Größe haben. Movie holt zunächst alle Rasterbilder in den Hauptspeicher und zeigt sie dann in schneller Folge. Es sollten nur soviele Rasterbilder angegeben werden, wie in den Hauptspeicher (ohne Swappen) passen.

Aufruf	`movie` *Dateiname*
Dateiformate	`SGI RGB-Image`

scope **SGI**

Interaktives Darstellen eines Rasterbildes, welches auch größer als 1280x1024 Pixels sein kann. Neben dem verkleinert dargestellten Gesamtbild wird noch ein vergrößerter Ausschnitt des Bildes gezeigt. Der Ausschnitt kann durch Verschieben eines Rechteckes über dem Gesamtbild verändert werden. Der Vergrößerungsfaktor wird über die Pfeiltasten eingestellt.

Aufruf	`scope` *Dateiname*
Dateiformate	`SGI RGB-Image`

getx11 [Utah Raster Toolkit] **Sun, SGI**

Darstellung eines Rasterbildes. Ist das Bild größer als der Bildschirm, kann mit der Maus der Mittelpunkt des Bildausschnittes verschoben werden ("Panning"). Mit der Maus kann auch interaktiv vergrößert oder verkleinert werden ("Zooming"). Mehrere Bilder können als Bildfolge ("Moviemode") oder als Einzelbilder dargestellt werden. Je nach Bildschirmtyp kann die Farbdarstellung auf 256 Farben beschränkt sein.

Aufruf	`getx11 [`*Optionen*`] [`*Dateiname(n)*`]`
Dateiformate	`RLE`

ghostscript — Sun, SGI

PostScript-Interpreter. Nach Interpretation der Eingabedatei können über die Tastatur weitere PostScript-Anweisungen eingegeben werden.

Aufruf	`gs [`*Optionen*`] [`*PostScript-Datei*`]`
Dateiformate	`PostScript`

ghostview — Sun, SGI

Ghostview ist eine interaktive Benutzungsoberfläche auf der Basis von X11 für den PostScript-Interpreter *ghostscript*.
Bem.: .eps-Dateien sollten mit der Zeichenfolge `%!PS-Adobe-2.0 EPSF-2.0` (nicht nur `%!`) eingeleitet werden.

Aufruf	`ghostview [`*Optionen*`] [`*PostScript-Datei*`]`
Dateiformate	`PostScript`

mpeg_play — Sun, SGI

Diese Version des MPEG-players der University of California, Berkeley, gibt einen MPEG I-Datenstrom über X11 aus.

Aufruf	`mpeg_play [`*Optionen*`] [`*MPEG-Dateiname*`]`
Dateiformate	`MPEG I`
Dokumentation	`/usr/local/X11/X11R5/man`

xdvi — Sun, SGI

Interaktive (Blättern, Zoomen) Darstellung einer .dvi-Datei.

Aufruf	`xdvi [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`DVI`

xmosaic — Sun, SGI

xmosaic ist ein 'World Wide Web'-Browser (WWW-Client) mit der Möglichkeit multimediale Informationen von anderen WWW-Servern zu holen und anzusehen bzw. anzuhören (Bild, Film und Ton).

Aufruf	`xmosaic`

xsee **Sun, SGI**

xsee ist ein Previewer für Rasterbilder im Targaformat.

Aufruf	`xsee` *Dateiname*
Dateiformate	`TARGA Typ 2, 10, 1 mit 24 oder 32 Bit/Pixel`

xv **Sun, SGI**

xv ist ein interaktiver Previewer für mehrere Rasterformate. Mit xv können Rasterbilder auch manipuliert (z.B. Farbänderungen, Ausschnitte, Größenänderungen) und in ein anderes Format gewandelt werden. Neben den o.a. Formaten kann noch nach PostScript gewandelt werden.

Aufruf	`xv [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`GIF, PBM, PGM, PPM, X11 bitmap, JPEG, RLE, Sun-Raster`

xwud **Sun, SGI**

Darstellen eines Rasterbildes.

Aufruf	`xwud [`*Optionen*`] [-in` *Dateiname*`]`
Dateiformate	`XWD (X11 Window Dump Format)`

4.2 Papier und Folie

4.2.1 Eingabe (Scanner)

Im ZIB steht derzeit nur ein Schwarz-Weiß-Scanner (Flachbett)[1] zur Verfügung. Es können A4-Vorlagen in Auflösungen von 100 bis 800 bpi gescannt werden. Mit Hilfe der zugehörigen Software (*PRESS View Color 2.0*) ist eine Bildnachbehandlung möglich. Die gescannten Daten werden als Rasterfile in folgenden Formaten gespeichert:

- Sun Raster
- TIFF (Tagged Image File Format)
- EPS (Encapsulated PostScript).

Die Daten können auch vektorisiert, also in Linienform verwandelt und dann in folgenden Formaten ausgegeben werden:

- EPS

[1] *FOCUS S800* der Firma Agfa-Gevaert

- DXF (kann von dem CAD-System *AutoCad* gelesen werden)
- PIC (Pixar Picture file)
- METAFONT (TEX).

PressView läuft unter OpenWindows, kann aber nur auf der Sun-Sparc-Station vispar7 gestartet werden, d.h., der Benutzer muß sich von seinem Rechner dort einloggen. Die Environmentvariable DISPLAY muß eingestellt sein und das xhost-Kommando für vispar7 (`xhost +vispar7`) gegeben sein.

Vor dem ersten Start von PressView muß mit *user.install* im eigenen home-Directory die Umgebung für die Scanner-Software eingerichtet worden sein (im Suchpfad muß `/usr/local/agfa/pressview/appl` stehen). Der Start erfolgt dann mit

```
% pressview
```

4.2.2 Ausgabe (Schwarzweißdrucker, Farbdrucker)

Schwarzweißdrucker

Im ZIB werden hauptsächlich zwei Typen von Laserdruckern und ein Leistungsdrucker verwendet:

Apple LaserWriter II (printername: `lw`),
Sun SPARCprinter (printername: `np`),
HP LaserJet IIISi (printername: `sl`)

Die Apple- und Sun-Printer drucken schwarz-weiß in 300 dpi Auflösung auf A4-Format (Papier oder Folie). Der HP-LaserJet steht im Rechnerraum und wird von Operateuren bedient. Dieser leistungsstarke Drucker (16 Seiten pro Minute) kann doppelseitig drucken und ist für große Ausgaben gedacht.

Die Drucker können über das Kommando

```
% lpr -Pprintername dateiname
```

zum Ausdrucken einer Datei veranlaßt werden. Mit

```
% lpq -Pprintername
```

kann man sich über den Status des Druckers und die zugehörige Print-Queue informieren.

Die Drucker interpretieren *PostScript Level 1*. Sind die ersten beiden Zeichen in der auszugebenden Datei *%!*, erkennt der Spooler, daß es sich um eine PostScript-Datei handelt. Im anderen Fall wird die Datei als ASCII-Text gewertet und in PostScript gewandelt.

Die *Apple LaserWriter* sind über serielle Schnittstellen angeschlossen, wodurch die Übertragungungsgeschwindigkeit verhältnismäßig gering ist. Bei der Ausgabe von großen Dateien ist dies zu berücksichtigen.

Mit dem Kommando

```
% /usr/local/transcript/enscript [optionen] dateiname
```

kann eine ASCII-Datei in PostScript gewandelt werden und jedes ausgebene Blatt mit einem Schriftkopf und einer Blattnumerierung versehen werden.

Farbdrucker

Derzeit steht dem ZIB der *Phaser III PXi* von Tektronix zur Verfügung (Raum 110b). Es ist ein Thermotransferdrucker, d.h., der Farbdruck wird durch das Aufbringen von kleinen Tintenwachstropfen in den Grundfarben *Magenta*, *Cyan*, *Gelb* und *Schwarz* erzeugt. Alle anderen Farben und Farbabstufungen werden durch Über- und Nebenbeinanderdrucken (Dithering) von Farbpunkten erzeugt (subtraktive Farbmischung bzw. Dithering). Auf diese Weise sind insgesamt 8 Millionen Farben möglich, wovon 8000 gleichzeitig dargestellt werden können. Durch das Nebeneinanderdrucken verschlechtert sich jedoch die räumliche Auflösung, so daß dünne Linien nicht in jeder möglichen Farbe ausgegeben werden können. Der Phaser interpretiert *PostScript (Level 1 und 2)* und HPGL. Da der Phaser einen eigenständigen Ethernet-Knoten darstellt, kann er von allen Workstations im Hause aus direkt angesprochen werden.

Als Ausgabemedium kann A4- und A3-Papier sowie A4-Spezial-Folie verwendet werden. Das Papier bzw. die Folie kann in einer Kassette liegen oder als Einzelblatt zugeführt werden.

Der Drucker kann über das Kommando

`% lpr -Pcp` *dateiname*	Zur Ausgabe von ASCII-PostScript-Dateien (Level 1 und 2).
`% lpr -Php` *dateiname*	Zur Ausgabe von HPGL-Dateien.

erreicht werden. Mit `lpq -Pcp` bzw. `lpq -Php` kann man sich über den Status des Druckers und die jeweilige Print-Queue informieren.

Im Normalfall steckt im Phaser die *leere* A4-Papierkassette, so daß der Benutzer erst ein Bild erhält, wenn er Papier oder Folie über den Einzelblatteinzug dem Drucker zuführt. A4-Einzelblätter sollten in der Regel quer, d.h. mit der breiten Seite, in den Papiereinzug gelegt werden. Die Papierkassette sollte immer leer bleiben, damit nicht versehentlich ganze Serien von Ausgaben auf den Phaser geschickt und dann gleich ausgegeben werden. Um A3-Ausgaben zu bekommen, muß die A3- Papierkassette anstelle der A4-Kassette in den Phaser gesteckt werden. Anschließend die Kassetten wieder austauschen!

ES KÜMMERE SICH ALSO JEDER UM SEINEN OUTPUT!

Die Kosten pro Seite sind erheblich höher als auf Schwarz-Weiß-Druckern. Bitte nur solche Bilder auf dem Phaser ausgeben, die sich als Farbbild wirklich lohnen.

Wenn der Phaser mehr als 2 Stunden nicht benutzt wurde, geht er in einen Standby-Modus über, den er wieder verläßt, wenn man eine Datei zur Ausgabe an den Drucker schickt oder eine Klappe am Drucker öffnet. Wird der Phaser mehr als 5 Tage nicht benutzt, kühlt er vollständig ab. Das Wachs zum Drucken

wird dann fest, und es dauert 15 - 20 Minuten bis der Drucker sich erwärmt hat und zu einer Ausgabe fähig ist.

Um die Farben dem gewünschten Ausgabemedium (Papier oder Folie) anzupassen, kann vor der eigentlichen bilderzeugenden PostScript-Datei eine spezielle PostScript-Datei zum Umschalten auf den Phaser gesendet werden. Mit der gleichen Technik läßt sich die Druckqualität einstellen. Folgende Vorschaltdateien sind vorhanden(unter serv04:/zib/ps):

`phsr3pap.ps`	Ausgabe auf Papier
`phsr3tra.ps`	Ausgabe auf Folie
`draft.ps`	Schwarz-Weiß-Ausgabe
`standard.ps`	Standard-Qualität (Default nach Einschalten der Geräte)
`enhanced.ps`	hohe Qualität, empfehlenswert
`premium.ps`	höchste mögliche Qualität, nur wenig besser als die vorherige, aber deutlich längere Druckzeit.

Eingestellte Qualitäten gelten bis zum Umschalten in eine andere Qualität, bzw. bis zum Ausschalten des Gerätes. Die ersten beiden Vorschaltdateien können mit den anderen kombiniert werden.

4.3 Fotografischer Film (Dia-Belichter)

Jedem Benutzer im ZIB steht über das Netz mit dem lpr-Kommando ein Diabelichter[2] zur Verfügung. Es können damit handelsübliche Filme für Kleinbildkameras (36x24mm) belichtet werden. Die maximale Auflösung beträgt 4096x2731 Pixel bei über 16 Millionen Farben. Der Diabelichter interpretiert drei Grafiksprachen:

Lasergraphics Language(LL):	wird im ZIB nicht benutzt.
Hewlett-Packard Graphics Language(HPGL):	geeignet für Vektorgrafik, z.B. zur Ausgabe von GRAZIL.
TARGA:	Raster-Format.

Der Diabelichter ist über einen PC mit dem Netz verbunden. Der PC übernimmt die Daten aus einem UNIX-Spoolbereich und leitet sie dann weiter an den Diabelichter. Da diese Geräte nicht immer eingeschaltet sind, wird der Spoolbereich mehrmals täglich nach Ausgabeaufträgen duchsucht. Sind Aufträge vorhanden und die Geräte nicht eingeschaltet, wird eine Mail an den Bediener abgesetzt.

Der Benutzer kann über das Kommando `lpr -Pdia_ps` folgende Formate an den Diabelichter übergeben:

PostScript (ps) und Encapsulated PostScript (eps)

Targa (tga)

[2] *Montage FR1* von der Firma PRESENTATION TECHNOLOGIES (Kanada).

HPGL (hpgl)

mehrere Raster-Formate (gif, hdp, mpnt, pbm, pic, pict, ppm, rgb, ras, rla, rle, tiff, xbm, xwd).

In Abhängigkeit von der Endung des Filenamens wird ein entsprechendes Konvertierungsprogramm durchlaufen, bevor die Ausgabe auf dem Diabelichter erfolgt. Bei unbekannter Endung wird PostScript angenommen.

Bei Ausgabe von Encapsulated PostScript wird durch die Bounding Box das Bild möglichst platzfüllend auf den Film gebracht (ohne Verzerrung).

Kommando dia

Für PostScript-Dateien steht das Kommando `dia` zur Verfügung; damit kann man sich das auszugebene Bild auf dem Bildschirm anzeigen lassen und durch Angabe von Optionen das Bild in der Größe verändern, drehen, verschieben, mit einer Hintergrund- bzw. Vordergrundfarbe versehen und das ZIB-Logo in die rechte obere Ecke integrieren. Die Voreinstellung für den Hintergrund ist weiß, für den Vordergrund schwarz.

Durch einen auf dem Bildschirm eingeblendeten Hilfsrahmen, eine dem Dia-Format entsprechende Bounding Box, wird die Anpassung des Bildes erleichtert. Der Rahmen wird nur auf dem Sichtgerät dargestellt.

Die Ausgabe der Bilder kann auf dem Diabelichter, PS-printer(lw), PS-colorprinter(cp) oder in eine Datei erfolgen. Da die durch das Kommando `dia` manipulierten PostScript-Dateien sehr groß werden können, kann die Environment-Variable `TMPDIR` auf ein Verzeichnis in einem Filesystem mit ausreichend Platz gesetzt werden. Voreingestellt ist `/tmp`.

Was ist beim Erstellen von Dias zu beachten?

Es ist ein Rand von ca. 5% der linearen Bildausdehnung von wichtiger Information frei zu halten, da dieser durch den Diarahmen abgedeckt werden kann. Dünne Linien sind zu vermeiden. Sie sind - besonders bei hellem Hintergrund - auf der Leinwand kaum sichtbar. Es sollte nicht zuviel Text auf einem Dia stehen und die Schrift sollte möglichst groß sein. Längere Texte also auf mehrere Dias verteilen! Viele Schrifttypen auf einem Bild sollte man vermeiden. Farbiger Hintergrund wird bei Textdias als angenehm empfunden (z.B. blauer Hintergrund mit weißer Schrift). Screendumps sollten, wenn möglich, vermieden werden, da durch die stark vergrößernde Projektion auf eine Leinwand die Pixelstruktur erkennbar wird.

Mit dem `dia`-Kommando kann bei PostScript-Bildern durch Skalieren (`-s 0.95 0.95`) ein freier Rand geschaffen werden (hierbei Hintergrund auf schwarz setzen, damit kein weißer Rand entsteht!) oder bei Textbildern der Vorder- bzw. Hintergrund verändert werden. Ebenfalls kann das ZIB-Logo in die rechte obere Ecke eingeblendet werden. Die zwei Abbildungen D.23 und D.24 im Anhang D auf Seite 129 zeigen unter Beachtung der obigen Empfehlungen erstellte Dias.

4.4 Video

Zur Aufzeichnung von Bewegtbildern steht im ZIB eine Video-Anlage zur Verfügung. Diese Anlage besteht aus folgenden Komponenten:

Laser Videodisc Recorder

Es handelt sich um einen Laser-Bildrecorder *LVR-6000A* der Firma Sony mit Steuergerät zum Aufzeichnen von Farbbildern und Ton auf eine einmal-beschreibbare, auswechselbare CRV-Platte („Bildplatte“) mit einer Kapazität von 36250 Bildern (d.h. 24 Minuten bewegte Bilder) pro Seite. Vertonung erfolgt, obwohl im Prinzip möglich, nicht auf der Bildplatte, sondern vorzugsweise auf dem Betacam-Videorecorder.

2 Videorecorder

Der Betacam-Recorder *PVW-2800P* von Sony ist ein Videorecorder für den professionellen Einsatz mit hoher Bildqualität. Er erlaubt Einzelbildaufzeichnung und auch das Ansteuern bzw. Suchen und Überschreiben einzelner Bilder. Eine Vertonung kann unabhängig von der Bildaufzeichnung zu einem späteren Zeitpunkt erfolgen. Auf diesem Recorder wird das Master-Band eines Videofilms erstellt.

Zum Erstellen von Kopien im VHS, bzw. S-VHS-Format ist noch ein weiterer Videorecorder (JVC *BR-S605*) verfügbar. Auf diesem Recorder wird nur das Master-Band kopiert, nicht geschnitten und keine Einzelbildaufzeichnung vorgenommen.

SGI-VideoCreator-Board

Der VideoCreator ist eine Zusatzkarte für Silicon Graphics-Rechner mit einem eigenen Framebuffer und analogem Video-RGB-Ausgang. Dieses Board stellt die (derzeit einzige) Verbindung im ZIB zwischen Rechnern und Videoapparatur dar.

Fernseh-Monitore

Für verschiedene Videoarbeiten, etwa das Zusammenstellen von Bildsequenzen von der Bildplatte auf Video, ist es notwendig, die Bildfolgen von zwei Geräten gleichzeitig zu beobachten. Dazu sind zwei Farbfernseh-Monitore vorhanden. Die Farben von Fernseh-Monitoren entsprechen nicht genau denen von Rechner-Bildschirmen. Auch deshalb sollte die Erstellung von Videofilmen über diese Monitore kontrolliert werden.

V-LAN-Transmitter/-Receiver

Über V-LAN-Transmitter/-Receiver[3] sind die Videorecorder und die Bildplatte von einem Silicon-Graphics-Rechner aus mittels entsprechender Software steuerbar.

Ablauf der Bildaufzeichnungen

[3] V-LAN ist ein herstellerneutrales Protokoll zur Steuerung von Videogeräten.

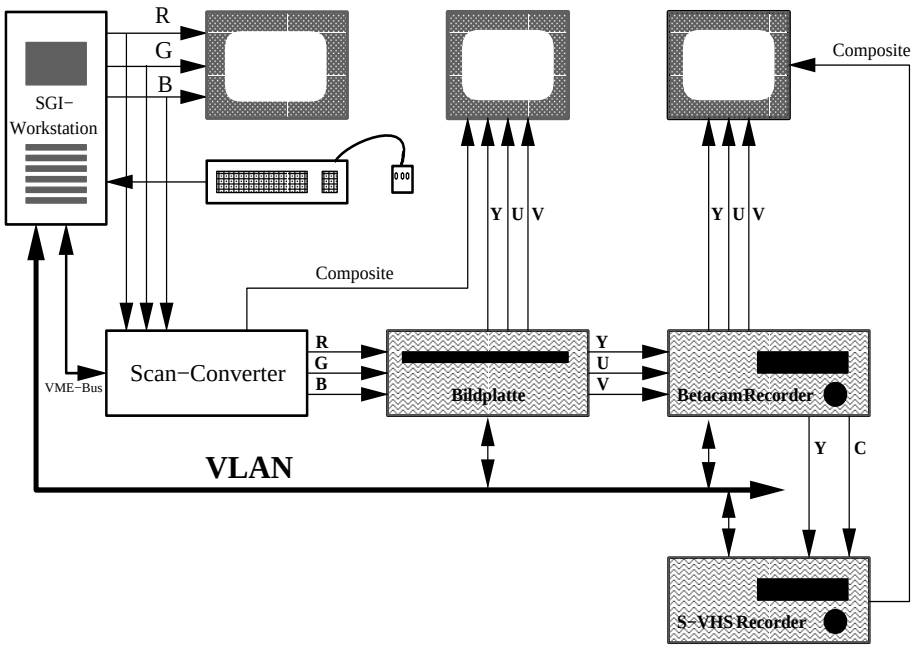

Abbildung 4.1: Videokonfiguration am ZIB

Die Bilddaten müssen als Rasterfiles in Silicon Graphics RGB-Image-Format vorliegen. Es kann von vielen Rasterformaten und auch von PostScript in das SGI RGB-Imageformat gewandelt werden (siehe Kap. 3.1.1). Die Bilder werden in den Framebuffer des VideoCreators geschrieben und von dort als RGB-Videosignale an die Bildplatte bzw. den Videorecorder übergeben und aufgezeichnet. Bei Bildserien ist es sinnvoll, die Bilder erst auf der Platte aufzuzeichnen, da der Videorecorder bei Einzelbildern, die nicht in Echtzeit aufgezeichnet werden können, für jedes Bild stoppen, zurücksetzen, neu beschleunigen und positionieren muß. Die damit verbundenen mechanischen Belastungen würden die Lebensdauer des Recorders wesentlich herabsetzen.

Postprocessing
Da das ZIB über keinen Video-Schnittplatz, Mischer, Effektgerät, Titelgenerator usw. verfügt, können Bilder und Bildsequenzen, die sich schon auf der Videoplatte befinden, nachträglich nicht mehr verändert werden. Es wurde beim Aufbau der Video-Anlage von vornherein berücksichtigt, daß sich die klassischen Postprocessing-Arbeiten im Rechner erledigen lassen - auch wenn dies heute noch nicht üblich ist. Am ZIB steht für diese Arbeiten der Video Composer (s. Kap. 2.3.8) zur Verfügung (nur auf der SGI-Workstation grafs11). Solange die Bilder noch als Rasterdatei vorliegen, können sie hiermit bearbeitet werden. Hier eine Liste der wichtigsten Bearbeitungsmöglichkeiten:

- Generierung von Texten, z.B. für Vorspann, Abspann oder Untertitel
- Ein-, Aus- und Überblenden von Bildern oder Bildsequenzen
- Videoeffekte, wie Wegschieben des vorherigen Bildes durch das neue

- Bildverarbeitung, z.B. Vergrößern, Verkleinern, Aufhellen, Farbveränderungen.

Diese Operationen können auch animiert ablaufen, die Zeitabhängigkeiten werden durch Parameter gesteuert. So können etwa die Positionen von Texten animiert werden, um diese über den Bildschirm laufen zu lassen. Möglichkeiten zur Vertonung sind z.Z. noch nicht am ZIB vorhanden, so daß auf externe Studios zurückgegriffen werden muß. Benutzer der ZIB-Rechenanlagen (auch externe) können einen Video-Aufzeichnungsservice nutzen, indem sie die Einzelbilder und eine Filmbeschreibung erstellen und dann die Aufzeichnung als Batch-Job starten. Eine genaue Beschreibung ist in [14] zu finden.

Was ist beim Erstellen von Bildern für Video zu beachten?

- Das Bild sollte eine Auflösung von 768x576 Pixel (Seitenverhältnis 4:3) haben.
- Es ist ein Rand von ca. 75 Pixel (je nach Bildschirm ca. 2 cm) von wichtiger Information frei zu halten, da dieser Rand bei der Wiedergabe auf Fernsehbildschirmen oder Videoprojektoren nicht sichtbar ist.
- Dünne Linien sind nach Möglichkeit zu vermeiden.
- Die Farben auf dem Workstation-Bildschirm weichen i.a. etwas von denen auf Videodisplays ab. Diese Bilder daher auf einem Fernsehmonitor kontrollieren!
- Wegen systembedingter Limitierungen bei der analogen Aufzeichnung von Video sollten keine reinen Farben hoher Intensität zur Darstellung verwendet werden. Die Kontraste zwischen nebeneinander liegenden Bildbereichen sollten ebenfalls nicht sehr hoch sein.
- Als Material für Videoaufzeichnungen können Bilder in allen Formaten verwendet werden, die sich mit entsprechender Software in das zur Aufzeichnung erforderliche SGI-RGB-Image-Format wandeln lassen (siehe Kap. 3.2).

Anhang A

Software-Liste

AGIL **Sun**

Pfad	`serv04:/zib/agil/alpha_test`

Environment	`setenv PHIGSDIR`
	`/zib/lib_sun4/phigs1.4/lib/phigs1.4/lib`
	`setenv AGIL_HOME`
	`/zib/agil/alpha_test/lib/sun4`

Dokumentation	`serv04:/zib/agil/alpha_test/doc`
Ansprechpartner	R. Wunderling

animate $\Longrightarrow$ Image Magick **Sun**

apE 2.0 **Sun, SGI**

Pfad (Sun) *Pfad (SIG)*	`/zib/apE/bin/sun4` `/zib/apE/bin/sgi`

Environment	`setenv APE_HOME /zib/apE`
	`setenv APE_BIN /zib/apE/bin`
	`setenv APE_HOST 'hostname'`

Dokumentation	`/zib/apE/doc/postscript`
Ansprechpartner	D. Stalling

Bob, Icol **SGI**

Pfad	`vispars1:/sgisoft/local/bin/bob` `vispars1:/sgisoft/local/bin/icol`
Dokumentation	`vispars1:/sgisoft/local/src/gvl/doc/allbob.*` `vispars1:/sgisoft/local/man`
Ansprechpartner	O. Paetsch

CLRpaint 1.01 **SGI**

Pfad	`vispars1:/sgisoft/local/bin/clrpaint`
Dokumentation	`vispars1:/sgisoft/local/doc/cp.doc.ps.gz`
Ansprechpartner	O. Paetsch

dia **Sun, SGI**

Pfad (Sun) *Pfad (SGI)*	`serv04:/zib/bin_sun4` `serv04:/zib/bin_sgi/sgi`
Environment	`setenv TMPDIR ...`
Dokumentation	`serv04:/zib/man/man1`
Ansprechpartner	H. Jaedicke

display $\Longrightarrow$ Image Magick **Sun**

Explorer 1.0 **Cray Y-MP**

Pfad	`cray:/usr/local/explorer`
Dokumentation	Cray Explorer Addendum / User's Guide APG-5510 1.0
Ansprechpartner	H.-H. Frese

Explorer 2.0 **SGI**

Pfad	`/usr/explorer/bin/explorer`
Environment	`setenv EXPLORERHOME /usr/explorer`
Dokumentation	`/usr/catman` Userguide: `/usr/explorer/doc/userguide`
Bemerkungen	großer Hauptspeicher (mind. 64 MB) & großer Swap space (mind. 150 MB) empfehlenswert
Ansprechpartner	T. Hoellerer, D. Stalling

FIGARO+ 3.0 **SGI**

Environment	mehr als 40 Variablen
Übersetzen/ *Binden*	mit Shell-Script `grafs11:/usr/figaro/libs/runfig`
Dokumentation	FIGARO+ 3.0 Dokumentation
Ansprechpartner	J. Langendorf

FrameMaker 3.1 **Sun**

Pfad	`$FMHOME/bin/imaker`
Environment	`setenv FMHOME /zib/frame_3.1`
Dokumentation	FrameMaker User's Guide Online-Hilfsfunktion zur Laufzeit `man framemaker` [26]
Ansprechpartner	W. Maibauer

FUGKS/NewGKS **Sun, Cray X-MP, Y-MP**

Environment (Sun)	`setenv GKSFNT /zib/etc/gks/gksfnt`
	`setenv GKSERR /zib/etc/gks/gkserr`
	`setenv GKSWDT /zib/etc/gks/gkswdt`

Binden (Sun)	`f77 <prog>.o -L/usr/local/zib_lib -lngks -lsuntool -lsunwindow -lpixrect -lm -o <prog>`

Environment (Cray)	`export GKSFNT /usr/local/etc/ngks/gksfnt`
	`export GKSERR /usr/local/etc/ngks/gkserr`
	`export GKSWDT /usr/local/etc/ngks/gkswdt`

Binden (Cray)	`segldr -o <prog> -l gks <prog>.o`
Dokumentation	Manpath: `serv04:/zib/man` Manpages: gks gks_new craygks metafile_new postscript sungraph hp7475
Ansprechpartner	J. Langendorf, O. Paetsch

get4d $\Longrightarrow$ Utah Raster Toolkit **SGI**

getsun $\Longrightarrow$ Utah Raster Toolkit **Sun**

getx11 $\Longrightarrow$ Utah Raster Toolkit **Sun, SGI**

ghostscript **Sun, SGI**

Pfad (Sun)	`/usr/local/zib_bin`
Pfad (SGI)	`vispars1:/sgisoft/local/bin`
Aufruf	`gs [`*Optionen*`] [`*PostScript-Datei*`]`
Dateiformate	`PostScript`
Dokumentation (Sun)	`serv04:/zib/man`
Dokumentation (SGI)	`vispars1:/sgisoft/local/man`

ghostview **Sun, SGI**

Pfad (Sun)	`serv04:/usr/local/zib_bin`
Pfad (SGI)	`vispars1:/sgisoft/local/bin`
Aufruf	`ghostview [`*Optionen*`] [`*PostScript-Datei*`]`
Dateiformate	`PostScript`
Dokumentation (Sun)	`serv04:/usr/local/X11/X11R5/man`
Dokumentation (SGI)	`vispars1:/sgisoft/local/man`

GNUPLOT 3.4 **Sun, SGI**

Pfad (Sun)	`serv04:/usr/local/zib_bin,`
Pfad (SGI)	`vispars1:/sgisoft/local/bin`
Dokumentation (Sun)	`serv04:/zib/man`
Dokumentation (SGI)	`vispars1:/sgisoft/local/man`
Ansprechpartner	D. Stalling, H.C. Hege

gplot, gplotaw, gplotm **Sun, SGI**

Pfad (Sun)	`serv04:/usr/local/zib_bin`
Pfad (SGI)	`vispars1:/sgisoft/local/bin`
Dokumentation (Sun)	`serv04:/zib/man`
Dokumentation (SGI)	`vispars1:/sgisoft/local/man`
Ansprechpartner	O. Paetsch

GRAPE **SGI**

Dokumentation	`vispars1:/vispar/vis/grape/doc`
Ansprechpartner	O. Paetsch

GRAZIL **Sun**

Pfad	`serv04:/zib/bin_sun4`

Environment	`setenv GKSFNT /usr/local/etc/ngks/gksfnt`
	`setenv GKSERR /usr/local/etc/ngks/gkserr`
	`setenv GKSWDT /usr/local/etc/ngks/gkswdt`

Dokumentation	`Manpath: serv04:/zib/man`
	`Manpages: grazil`
	`GRAZIL Handbuch [38]`

Ansprechpartner	`J. Langendorf`

GRAZIL3D **Sun**

Pfad	`serv04:/zib/bin_sun4`

Environment	`setenv PHIGSHOME /zib/phigs2.0_pl14`
	`setenv XGLHOME $PHIGSHOME/lib/phigs2.0`
	`setenv LD_LIBRARY_PATH`
	`$PHIGSHOME/lib:$XGLHOME/lib:$LD_LIBRARY_PATH`

Dokumentation	`Manpath: serv04:/zib/man`
	`Manpages: grazil3d`
	`GRAZIL3D Handbuch`

Ansprechpartner	`J. Langendorf`

Image Magick **Sun**

Pfad	`/usr/bin/X11`
Dokumentation	`/usr/local/X11/X11R5/man`
Ansprechpartner	K. Peter

imf_dspl **SGI**

Pfad	`/usr/local/wave/TAV3.0/bin`
Aufruf	`imf_dspl [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`Wavefront RLA, TIFF, TARGA, SGI RGB-Image, ...`
Dokumentation	`Handbuch Advanced Visualizer`
Besonderheiten	Es besteht nur eine Lizenz auf grafs11.

imgsnap **SGI**

Pfad	`/usr/sbin`
Aufruf	`imgsnap [`*Dateiname*`]`
erzeugte(s) Dateiformat(e)	`SGI RGB-Image, TIFF, FIT`
Dokumentation	`/usr/catman`

imgview **SGI**

Pfad	`/usr/sbin`
Aufruf	`imgview [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`SGI RGB-Image, TIFF, FIT`
Dokumentation	`/usr/catman`

Iris Inventor **SGI**

Übersetzen/	C++: mit SGI CC-Compiler, bei Vers. 3.0: Flag -v2 C: bel. ANSI-Compiler.
Binden	Libraries s. Dokumentation.
Dokumentation	Iris Inventor Programming Guide
Ansprechpartner	T. Hoellerer, D. Stalling

ipaste **SGI**

Pfad	`/usr/sbin`
Aufruf	`ipaste [`*Optionen*`]` *Dateiname*
Dateiformate	`SGI RGB-Image`
Dokumentation	`/usr/catman`

Iris Raster Tools **SGI**

Pfad	`/usr/sbin`
Dokumentation	`/usr/catman`
Ansprechpartner	D. Stalling, O. Paetsch

MiniGraphik **Sun, Macintosh**

Pfad	`newton:/home/neville/MiniGraphik/sun-4`
Dokumentation	`newton:/home/neville/MiniGraphik/doc/`
Ansprechpartner	R. Roitzsch, A.Wendt

movie **SGI**

Pfad	`/usr/sbin`
Aufruf	`movie` *Dateiname*
Dateiformate	`SGI RGB-Image`
Dokumentation	`/usr/catman`

MPGS 5.0 **SGI, Cray**

läuft auf	`aventurin, cray`
Aufruf	`mpgb5`
Dokumentation	`doc -l mpgs5`
Ansprechpartner	H.-H. Frese

NAG Graphics Library Mark 3 **Sun, Cray**

läuft auf	crax, cray, ufer
Übersetzen/Binden mit PostScript (crax,cray)	`cf77 <prog>.f -lnaggl -lnagaps -lnag`
Übersetzen/Binden mit PostScript (ufer)	`f77 -fast <prog>.f -lnaggl -lnagaps -lnag`
Übersetzen/Binden mit GKS (crax, cray)	`cf77 <prog>.f -lnaggl -lnaggks -lnag -lgks`
Übersetzen/Binden mit GKS (ufer)	`f77 -fast <prog>.f -lnaggl -lnaggks -lnag -lgks`
Environment	`GKSWKT=<workstationtyp>; export GKSWKT`
Online-Help (ufer)	`naghelp`
Dokumentation (ufer)	`doc -l naggl`
Ansprechpartner	H.-H. Frese

NCAR Graphics **Sun, SGI, Cray**

läuft auf	aventurin, crax, cray, ufer
Pfad	`/usr/local/ncarg`
Übersetzen/Binden	`nacrg f77 <prog>.f`
Dokumentation *Dokumentation (ufer)*	`man ncarg` `doc -l ncarg`
Ansprechpartner	H.-H. Frese

pageview **Sun**

Besonderheiten	Nur auf Sun's unter OpenWindow
Pfad	`/usr/openwin/bin`
Aufruf	`pageview [`*Optionen*`] [`*PostScript-Datei*`]`
Dateiformate	`PostScript`
Dokumentation	`/usr/openwin/man`

PBMPLUS Package **Sun, SGI**

Pfad (Sun)	`serv04:/zib/pbmplus/pbmplus-sun4/bin`
Pfad (SGI)	`vispars1:/sgisoft/local/bin`
Dokumentation (Sun)	`serv04:/zib/pbmplus/man`
Dokumentation (SGI)	`vispars1:/sgisoft/local/man`
Ansprechpartner	D. Stalling, O. Paetsch, K. Peter

PEX-SI **Sun**

Environment	`setenv LD_LIBRARY_PATH /usr/lib:/usr/openwin/lib`
Übersetzen/ Binden	`cc -I/usr/local/X11/X11R5-sun4/include/X11 <prog>.c -L/usr/local/X11/X11R5-sun4/lib -lphigs -lX11 -lm -o <prog>`
Dokumentation	`setenv X11R5_PATH /usr/local/X11/X11R5-sun4 $X11R5_PATH/doc/extensions/PEX/SI/*`
Ansprechpartner	J. Langendorf, O. Paetsch

PressView **Sun**

läuft nur auf	Sun-sparc vispar7
Pfad	`vispar7:/usr/local/agfa/pressview/appl`
Environment	`wird mit` *Pfad/user.install* `eingestellt`
Ansprechpartner	H. Jaedicke

Rayshade **Sun, SGI**

Pfad	`/usr/local/zib_bin/rayshade`
Dokumentation	`vispars1:/vispar/vis/rayshade/Doc`
Ansprechpartner	O. Paetsch, H. C. Hege

SciAn **SGI**

Pfad	`vispars1:/sgisoft/local/bin/scian`
Dokumentation	`vispars1:/sgisoft/local/src/SciAn/doc/*` Online-Hilfsfunktion zur Laufzeit
Ansprechpartner	O. Paetsch

scope **SGI**

Pfad	`/usr/sbin`
Aufruf	`scope` *Dateiname*
Dateiformate	`SGI RGB-Image`
Dokumentation	`/usr/catman`

screendump **Sun**

Pfad	`/bin`
Aufruf	`screendump [`*Optionen*`] [`*Dateiname*`]`
erzeugte(s) Dateiformat(e)	`Sun-Raster`
Dokumentation	`/usr/man`

screenload **Sun**

Pfad	`/bin`
Aufruf	`screenload [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`Sun-Raster`
Dokumentation	`/usr/man`

scrsave **SGI**

Pfad	`/usr/sbin`
Aufruf	`scrsave` *Dateiname* `[`*X-, Y-Koordinaten*`] [-b]`
erzeugte(s) Dateiformat(e)	`SGI RGB-Image`
Dokumentation	`/usr/catman`

SDSC Image Tools 2.0 **Sun, SGI, Cray**

Pfad (Sun)
Pfad (SGI)
Pfad (Cray)
	`vispar:/vispar/vis/sdsc2.1/bin-sun4`
	`vispars1:/sgisoft/local/bin`
	`.../bin`

Dokumentation (Sun)
Dokumentation (SGI)
Dokumentation (Cray)
	`vispar:/vispar/vis/sdsc2.1/man`
	`vispars1:/sgisoft/local/man`
	`.../man`

Ansprechpartner	D. Stalling

Showcase **SGI**

Pfad	`/usr/sbin/showcase`
Dokumentation	IRIS Showcase User's Guide Online-Hilfsfunktion zur Laufzeit
Ansprechpartner	D. Stalling, O. Paetsch

snapshot **SGI**

Pfad	`/usr/sbin`
Aufruf	`snapshot [-b]`
erzeugte(s) Dateiformat(e)	`SGI RGB-Image`
Dokumentation	`/usr/catman`

snapshot **Sun**

Besonderheiten	Nur auf Sun's unter OpenWindow
Pfad	`/usr/openwin/bin`
Aufruf	`snapshot [`*Optionen*`]`
erzeugte(s) Dateiformat(e)	`Sun-Raster`
Dokumentation	`/usr/openwin/man`

SunPHIGS 2.0 **Sun**

Environment	`setenv PHIGSHOME /zib/phigs2.0/lib/phigs2.0`
	`setenv LD_LIBRARY_PATH`
	`$PHIGSHOME/lib:$LD_LIBRARY_PATH`

Übersetzen/
 	
Binden
	`cc -I/zib/phigs2.0/include`
	`-I/usr/openwin/include <prog>.c -lphigs`
	`-lxgl -lX11 -lm -o <prog>`
	`f77 -I/zib/phigs2.0/include <prog>.F`
	`-L/zib/phigs2.0/lib -lphigs77 -llphigs`
	`-lphigs -lxgl -lX11 -lm -o <prog>`

Dokumentation	SunPHIGS Dokumentation
Ansprechpartner	J. Langendorf, O. Paetsch

TEX, LATEX, und `dvips` mit `psfig.sty` **Sun, SGI**

Pfad	`serv04:/zib/TeX3.141/bin-$ARCH` und `serv04:/zib/TeX3.141/lib/tex/inputs/psfig`

Dokumentation	`serv04:/zib/TeX3.141/doc/dvips`
	`serv04:/zib/TeX3.141/doc/psfig`
	`serv04:/zib/TeX3.141/doc/psfrag`

Ansprechpartner	TEX-Beratung ist nicht vorgesehen. Alle mit der Installation zusammenhängenden Fragen per E-Mail an `texmaint` richten.

UniChem 2.0 **SGI, CRAY Y-MP**

läuft auf	aventurin, cray
Pfad	`/usr/local/unichem`
Aufruf	`unichem`
Dokumentation (ufer)	`doc -l unichem`
Ansprechpartner	Dr. T. Steinke

Utah Raster Toolkit **SGI**

Pfad	`vispars1:/sgisoft/local/bin`
Dokumentation	`vispars1:/sgisoft/local/man`
Ansprechpartner	D. Stalling, O. Paetsch

X11, Xt und Widget-Sets **Sun, SGI**

läuft auf	X11, Xt, Xaw: Sun, SGI Motif: (bisher nur) SGI OpenLook: Sun
Pfad (Sun) *Pfad (SGI)*	`/usr/local/X11` `/usr/lib/X11`
Dokumentation (Sun) *Dokumentation (SGI)*	`/usr/local/X11/X11R5/man` `/usr/catman`
Ansprechpartner	K. Peter

xdvi **Sun, SGI**

Pfad (Sun) *Pfad (SGI)*	`/zib/TeX3.141/bin-sun4` `/zib/TeX3.141/bin-iris`
Aufruf	`xdvi [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`DVI`
Dokumentation	`/zib/TeX3.141/man`

xfig **Sun, SGI**

Pfad	`/usr/bin/X11/xfig`

Dokumentation (Sun)
Dokumentation (SGI)
	`serv04:/zib/man_X11R4/mann`
	`/usr/catman`
	Manpages: `xfig fig2dev`

Ansprechpartner	K. Peter

XGKS **Sun, Cray**

läuft auf	Sun, Cray X-MP und Y-MP
Environment (Sun)	`setenv XGKSFontDir` `/zib/etc/gks/xgks/xgksfonts`

Binden (Sun)	`f77 <prog>.o -L/usr/local/zib_lib -lfxgks`
	`-lxgks -lX11 -lm -o <prog>`
	`cc <prog>.o -L/usr/local/zib_lib -lxgks`
	`-lX11 -lm -o <prog>`

Environment (Cray)	`export XGKSFontDir=/usr/lib/xgksfonts`
Übersetzen/ *Binden (Cray)*	`cf77 -l xgks,X11 -o <prog> <prog>.f` `cc -l xgks,X11 -o program program.c`
Dokumentation	`serv04:/zib/etc/gks/xgks/userdoc/xgks.doc`
Ansprechpartner	H. Jaedicke, O. Paetsch

xgrab, xgrabsc **Sun**

Pfad	`/usr/bin/X11`
Aufruf	`xgrab` `xgrabsc [`*Optionen*`]`
erzeugte(s) Dateiformat(e)	`XWD, X Bitmap, X Pixmap, PostScript, Encapsulated PostScript`
Dokumentation	`/usr/local/X11/X11R5/man`

Xgraphic **Sun**

Pfad	`/zib/Xgraphic/`
Dokumentation	`/zib/Xgraphic/doc.tex`
Ansprechpartner	A. Hohmann, K. Gatermann

xloadimage, xli, xview, xsetbg, xlito **Sun**

Pfad	`/usr/bin/X11`

Aufruf	`xloadimage [`*Optionen*`] [`*Dateiname(n)*`],`
	`xli [`*Optionen*`] [`*Dateiname(n)*`],`
	`xview [`*Optionen*`] [`*Dateiname(n)*`],`
	`xsetbg [`*Optionen*`] [`*Dateiname(n)*`],`
	`xlito [`*Optionen*`] [`*Dateiname(n)*`]`

Dateiformate	`FBM, GEM, GIF, G3 FAX, JFIF, MacPaint, PCX, PBM, PGM, PPM, Sun Raster, RLE, X Pixmap, X11 Bitmap, XWD, *.Z`
Dokumentation	`/usr/local/X11/X11R5/man`

xmosaic **Sun, SGI**

Pfad (Sun) *Pfad (SGI)*	`/usr/bin/X11` `vispars1:/sgisoft/local/bin`
Aufruf	`xmosaic`

XPaint **Sun, SGI**

Pfad (Sun) *Pfad (SGI)*	`/usr/bin/X11` `vispars1:/sgisoft/local/bin`
Dokumentation (Sun) *Dokumentation (SGI)*	`/usr/local/X11/X11R5/man` `vispars1:/sgisoft/local/man/xpaint`
Ansprechpartner	K. Peter, O. Paetsch

xsee **Sun, SGI**

Pfad(Sun) *Pfad(SGI)*	`vispars1:/vispar/vis/bin-sun4` `vispars1:/vispar/vis/bin-sgi`
Aufruf	`xsee` *Dateiname*
Dateiformate	`TARGA Typ 2, 10, 1 mit 24 oder 32 Bit/Pixel`

xtex **Sun**

Pfad	`/usr/bin/X11`
Aufruf	`xtex [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`DVI`
Dokumentation	`/usr/local/X11/X11R5/man`

xv **Sun, SGI**

Pfad (Sun) *Pfad (SGI)*	`/usr/bin/X11` `vispars1:/sgisoft/local/bin`
Aufruf	`xv [`*Optionen*`] [`*Dateiname*`]`
Dateiformate	`GIF, PBM, PGM, PPM, X11 bitmap, JPEG, RLE, Sun-Raster`
Dokumentation (Sun) *Dokumentation (SGI)*	`/usr/local/X11/X11R5/man` `vispars1:/sgisoft/local/man`

xwd **Sun, SGI**

Pfad	`/usr/bin/X11`
Aufruf	`xwd [`*Optionen*`]`
erzeugte(s) Dateiformat(e)	`XWD (X11 Window Dump Format)`
Dokumentation (Sun) *Dokumentation (SGI)*	`/usr/local/X11/X11R5/man` `/usr/catman`

xwud **Sun, SGI**

Pfad	`/usr/bin/X11`
Aufruf	`xwud [`*Optionen*`] [-in` *Dateiname*`]`
Dateiformate	`XWD (X11 Window Dump Format)`
Dokumentation (Sun) *Dokumentation (SGI)*	`/usr/local/X11/X11R5/man` `/usr/catman`

Anhang B

Glossar

Aliasing. Der Begriff stammt aus der Signalverarbeitung. Wird ein Signal zu grob abgetastet, so können hochfrequente Anteile des Originalsignals, die oberhalb der sogenannten Nyquist-Frequenz liegen, im resultierenden Signal als niedrigfrequente Komponenten erscheinen. Die hohen Frequenzen führen somit zu einer „falschen Identität“, einem *Aliasing*, der niedrigfrequenten Anteile. Gegenmaßnahmen sind unter dem Begriff →*Anti-Aliasing* bekannt.

In der Computergrafik werden alle Artefakte, die durch Digitalisierung analoger Signale oder durch Diskretisierung der Definitionsbereiche zu berechnender Funktionen entstehen, als Aliasing bezeichnet.

Bei der Bildsynthese werden die drei Kontinua Raum, Zeit und Farbe diskretisiert. Entsprechend treten *räumliches, zeitliches und farbliches Aliasing* auf. Artefakte durch räumliches Aliasing sind stufige Kanten (*jagging*), unregelmäßig berandete Glanzlichter und fehlende Details. Zeitliches Aliasing äußert sich in ruckhaften Bewegungen und Flicker-Effekten. Farbliches Aliasing tritt auf, wenn bei Intensitätsberechnungen statt des gesamten sichtbaren Spektrums nur wenige Punkte (typischerweise nur drei Grundfarben) berücksichtigt werden. Es führt zu falschen Farbwerten und Intensitäten.

Alpha-Kanal. Eine Bilddatei mit *Alpha-Kanal* (alpha channel) enthält für jedes →Pixel einen zusätzlichen Wert, der die Transparenz angibt. Dieser Wert wird vor allem dann verwendet, wenn aus zwei Bildern ein neues aufgebaut werden soll. Wenn Bild A über ein Hintergrundbild B gelegt werden soll und einige Pixel von A als transparent markiert sind, scheint an diesen Stellen der Hintergrund B durch. Alpha-Werte sind in der Regel ganze Zahlen zwischen 0 und 255. Der Wert 0 bedeutet transparent und 255 undurchsichtig (engl. opaque). Bei Zwischenwerten wird die resultierende Farbe aus den entsprechend gewichteten Farben von Vordergrund und Hintergrund berechnet. Siehe auch →Kanal.

Ambient Light. Siehe →Beleuchtungsmodell.

Anti-Aliasing. Maßnahmen zur Verminderung von →*Aliasing* bezeichnet man als *Anti-Aliasing*. Generell wird eine Verfeinerung der Diskretisierung in der Computergrafik auch Oversampling oder Supersampling genannt. So kann man räumlichem Aliasing begegnen, indem man ein Rasterbild in größerer Auflösung berechnet und dann mehrere →Pixel geeignet zu einem neuen zusammenfaßt. Sind die Funktionsverläufe analytisch bekannt, so läßt sich dies auch nutzen. Beispielsweise kann in einem Rasterbild die Stufigkeit von analytisch bekannten Linien vermindert werden, indem Pixeln Farbwerte entsprechend dem Bedeckungsgrad zugeordnet werden. Zeitlichem Aliasing kann man durch Verwischen von Bewegungen (motion blur) begegnen.

Beleuchtungsmodell. Unter einem *Beleuchtungsmodell* (engl. illumination model, lighting model oder shading model) versteht man die Gesamtheit der bei der Berechnung (→Rendering) von Farbwerten und Intensitäten auf Oberflächen und im Inneren von geometrischen Objekten angewandten Regeln und Gleichungen.

Den heutzutage verwendeten Beleuchtungsmodellen liegt eine mehr oder weniger physikalische Betrachtungsweise zugrunde: Lichtstrahlen erfahren Reflektion, Absorption und Brechung an Oberflächen und werden beim Durchqueren von Volumina gedämpft. Die Vorgänge sind abhängig von Position und Abstrahlungscharakteristiken der Lichtquellen, von der Farbe und Intensität des einfallenden (evtl. schon mehrfach gestreuten) Lichtstrahls, von den Materialeigenschaften des betreffenden Objekts und von der Richtung der weiterverfolgten, gestreuten Strahlen. Die ersten in die Szene fallenden Strahlen sind definiert durch Position und Abstrahlungscharakteristiken der Lichtquellen; von den aus der Szene gestreuten Strahlen interessieren diejenigen, die das Auge des Beobachter erreichen.

In Beleuchtungsmodellen wird auch von unphysikalischen Parametern und Gleichungen Gebrauch gemacht, teils um gewisse Charakteristika der abzubildenden Szene visuell hervorzuheben, teils um strukturellen Mängeln von Bildberechnungsverfahren zu begegnen. So wird – u.a. beim →Raytracing – die nicht berücksichtigte diffuse Reflexion zwischen Objekten, welche zu einer Aufhellung der Szene führt, durch ein künstliches Umgebungslicht (→Ambient Light) simuliert.

Bildkompression. Rasterbilder mit einer Seitenlänge von typischerweise einigen hundert bis einigen tausend →Pixeln benötigen viel Speicherplatz, insbesondere wenn sie mehrere Farbkanäle und einen →*Alpha-Kanal* enthalten. Deshalb bieten viele Bildformate Möglichkeiten zur Datenkompression an. Man unterscheidet *verlustfreie* und *verlustbehaftete* Kompressionsververfahren (compression schemes). Für verlustbehaftete Verfahren, die zu erheblich besseren Kompressionsraten führen, wurden in jüngster Zeit wegen der wachsenden Bedeutung für die Unterhaltungselektronik Standards (MPEG, JPEG[1]) definiert. Für die Anwendung im

[1] JPEG enthält neben verlustbehafteten Verfahren auch verlustfreie.

wissenschaftlichen Rechnen sind verlustbehaftete Verfahren jedoch weniger geeignet, da sie zu unkontrollierten und bei computererzeugten Bildern teilweise unannehmbar starken Verfälschungen führen.

Das mit Abstand am häufigsten angewandte verlustfreie Verfahren ist die →*Lauflängen-Kodierung*. Aber auch außerhalb der Bildverarbeitung anwendbare Algorithmen wie *LZW-Codierung* oder *Huffman-Codierung* werden genutzt. Diese Algorithmen liefern bei Anwendung auf Bilddaten oft erheblich bessere Ergebnisse, wenn man vor der Kompression die Farbwerte durch Farbdifferenzen zwischen benachbarten Pixeln in benachbarten Zeilen des Bildes ersetzt.

Bildspeicher. In einem Bildspeicher (*Frame-Buffer* oder *Bildwiederholspeicher*) werden Bilder in Form von Pixelwerten abgelegt – meist, um sie für die Ausgabe, etwa auf einem Bildschirm, bereit zu halten. Zur Darstellung auf einem Bildschirm wird das Bild mit einer bestimmten Bildwiederholfrequenz (heute 50 - 100 Hz, typisch 70 Hz) aus dem Bildspeicher gelesen und in ein entsprechendes Analogsignal gewandelt.

In modernen Grafik-Workstations findet die Bildberechnung oft nicht im Hauptspeicher, sondern direkt auf dem Bildspeicher statt, insbesondere wenn spezielle Grafik-Hardware genutzt wird. Neben den für die Speicherung von Pixelwerten erforderlichen →Bit-Ebenen treten dann noch weitere Bit-Ebenen hinzu.

Bit-Ebene. Zu jedem auf einem Bildschirm darzustellenden →Pixel korrespondiert eine Speicherzelle im Frame-Buffer. Die Bits an n-ter Stelle der Speicherzellen stellen die n-te *Bit-Ebene* (*bit plane*) dar. Typisch ausgelegte Frame-Buffer enthalten heute *Image-Bit-Planes* für die Farbwerte, *Alpha-Planes* für Transparenzwerte, *Overlay-* und *Underlay-Planes* zur Verwendung durch den Window-Manager, etwa für Pop-Up-Menus, und *Window-Planes* zur Maskierung von Pixeln gegen Updates und zur Speicherung von Darstellungscharakteristiken (etwa Single-Buffering versus →Double-Buffering oder RGB versus Farbindex, s. →Farbtabelle).

Typische Auslegungen sind:

Image-Bit-Planes: für Schwarz-Weiß: 2, für Grauwerte oder für Farbtabellen-Indizes 8-12, für RGB-Werte 12-36 (→True Color: ≥ 24); →Double-Buffering erfordert eine Verdoppelung dieser Werte.

Alpha-Planes: normal 8, maximal 12.

Overlay-/Underlay-Planes: 4.

Window-Planes: 4-8.

In modernen Grafik-Workstations geschieht die hardware-unterstützte Bildberechnung direkt auf dem Frame-Buffer. Zur Unterstützung der Algorithmen werden dafür weitere Bit-Planes vorgesehen: *Depth-Planes* als

Z-Buffer, *Stencil-Planes* zur Kennzeichnung von Pixeln, etwa für priorisierende Algorithmen, und *Texture-Planes* zur Speicherung von Textur-Elementen, bestehend aus RGB- und Alpha-Werten. In besonders gut ausgestatteten Grafik-Rechnern sind zur Unterstützung weiterer Funktionen wie etwa Mischen, Überblenden, Filtern und Falten sogenannte *Accumulation-Planes* vorgesehen, auf denen sich allgemeine arithmetische und logische Operationen ausführen lassen.

Typische Auslegungen sind:

Depth-Planes: 24, manchmal auch 32.

Stencil-Planes: 8.

Texture-Planes: 32, maximal 64.

Accumulation-Planes: 16-48.

Channel, siehe →Kanal.

Bitmap. In einer *Bitmap* steht für jedes →Pixel nur ein einziges Bit zur Speicherung der Farbinformation zur Verfügung. Reine Schwarzweißbilder und Schriften (*Fonts*) werden oft in Form einer Bitmap gespeichert.

CMY, siehe →Farbmodell.

CSG, Constructive Solid Geometry, bezeichnet eine Modelliertechnik, bei der geometrische Basis-Objekte mit den booleschen Mengenoperationen (Vereinigung, Subtraktion und Differenz) kombiniert werden, um Körper komplexerer Form zu bilden. Derart aufgebaute Körper werden intern durch einen Baum repräsentiert (Knoten = Mengenoperatoren, Endknoten = Basisobjekte).

Colorimetrie Die Bemühungen der Farbmessung (*Colorimetrie*) gehen dahin, eine einheitliche genormte Repräsentation von Farben zu erarbeiten, die eineindeutig einen empfundenen Farbeindruck auf ein Zahlentripel abbildet. Dazu führte die *Commission Internationale de l'Eclairage* (CIE) um 1930 mit ca. 40 normal farbtüchtigen Versuchspersonen eine Reihe von Color-Matching Experimenten (→Farbwahrnehmung) durch, um einen sogenannten „Standardbeobachter“ zu schaffen.

Mit drei realen Grundfarben (sogenannten *Primaries*), nämlich einem reinen, d.h. monospektralen Rot, Blau und Grün, sollten nacheinander im Abstand von ca. 10 nm alle reinen Farben über dem gesamten sichtbaren Spektrum abgeglichen werden. Die Ausgleichskurven über die protokollierten Wichtungen der drei Primaries aller Versuchspersonen ergaben die genormten Color-Matching-Funktionen für die drei gewählten Primaries. Color-Matching-Funktionen stellen also die Wichtungsfaktoren für je ein Primary der Wellenlänge dar. Sie geben Auskunft darüber, welche Anteile von den drei Primaries zusammengemischt werden müssen, um ein von

monospektralen Licht der jeweiligen Wellenlänge visuell nicht zu unterscheidendes Licht erhalten.

Wenn man als Grundfarben drei Farben aus dem sichtbaren Spektrum wählt, dann kann man mit nur positiven Gewichtungen nur einen Teilbereich (ein sog. *Gamut*) aller sichtbaren Farben zusammenmischen. Solche drei sichtbaren Primaries haben also korrespondierende Color-Matching-Funktionen, die zum Teil negative Werte annehmen, was sowohl vom mathematischen als auch vom Standpunkt einer konsistenten Definition unerwünscht ist.

Der „Trick" zur Umgehung dieses Problems besteht darin, daß die schließlich zugrunde gelegten Primaries **X**, **Y** und **Z** keine sichtbaren Farben repräsentieren. Sie sind nur über ihre Color-Matching-Funktionen ($\bar{x}, \bar{y}$ und $\bar{z}$) definiert, die wiederum durch eine einfache affine Abbildung aus den oben beschriebenen Experimentaldaten bestimmt wurden (wobei genug Freiheiten für die Verwendung einer biologisch relevanten Abbildung [2] als Funktion $\bar{y}$ vorhanden waren).

1931 wurden ($\bar{x}, \bar{y}$ und $\bar{z}$) auf dieser Grundlage in Abständen von 1 nm tabelliert. Diese Norm ist bis heute in der Computergrafik relevant. Allerdings hat sich 1976 eine Verbesserung ergeben: der CIE-LUV uniforme Farbraum. In diesem gilt dann zusätzlich, daß gleiche Abstände in dem zungenförmigen Chromaticity-Diagramm (s.u.) auch gleichen perzeptuellen Farbabständen entsprechen.

Chromatizitätswerte im gleichnamigen Diagramm hängen nur von *hue* und *saturation* ab (nicht von *radiance*[3]), da zugunsten einer zweidimensionalen Darstellung nur die Schnittebene $X + Y + Z = 1$ betrachtet wird (X, Y, Z sind die Anteile eines Farbeindrucks an den Primaries **X**, **Y** und **Z**), was gleichbedeutend ist mit einer Normierung bezüglich des Gesamtbetrages an Lichtenergie $X + Y + Z$.

Viele Sachverhalte der Farbentheorie lassen sich auf einfache Weise am CIE-Chromatizitätsdigramm veranschaulichen, so z.B. alle farbbeschreibenden Begriffe aus der Colorimetrie. Für eine gelungene Beschreibung dieses Diagrams siehe [19], S. 579-584.

Color Quantization, siehe →Halftoning.

Compression Schemes, siehe →Bildkompression.

Depth Cueing. Die Entfernung eines Objektes vom Betrachter kann durch die Farbintensität, mit der es dargestellt wird, verdeutlicht werden. Aufgrund der atmosphärischen Absorption erscheinen weit entfernte meist dunkler als nahe. Das sogenannte *depth cueing* in der Computergrafik approximiert diesen Effekt auf einfachste Art und Weise. Meist werden zwei

[2] *Luminous-Efficiency-Function*, die die Sensitivität des menschlichen Auges auf Licht konstanter Energie in Abhängigkeit von der dominanten Wellenlänge beschreibt

[3] Zu unterscheiden sind *radiance*, die den Gesamtenergiebetrag einer Lichterscheinung beschreibt, und *luminance*, der Betrag der vom Auge aufgenommenen Energie.

parallele Ebenen definiert, die sogenannten *depth cueing planes*. Mit jeder Ebene ist ein Skalierungsfaktor assoziiert, der das Verhältnis von Objektfarbe und Hintergrundfarbe (*depth cue color*) bestimmt. Zwischen den Ebenen kann der Skalierungsfaktor durch lineare Interpolation bestimmt werden. Wenig entfernte Objekte erscheinen in ihrer natürlichen Farbe, weit entfernte werden dagegen meist einheitlich grau dargestellt.

Dithering. Durch sogenanntes *Dithering* kann man die Zahl der möglichen Grau- oder Farbstufen auf Kosten der räumlichen Auflösung erhöhen, indem man Grau- bzw. Farbwerte durch geeignet gemusterte Rechtecke aus mehreren Rasterpunkten „emuliert“. Vgl. dazu auch →Halftoning.

DVI-Treiber sind Programme, mit denen `.dvi`-Dateien auf verschiedene Ausgabegeräte wie Drucker oder Bildschirm gegeben werden können. Die Treiber unterscheiden sich hinsichtlich der angesprochenen Geräte und den unterstützten `special`-Makros. `.dvi`-Dateien werden von TEX oder einem seiner Derivate erzeugt.

Double-Buffering. Um auf dem Bildschirm eine fließende Darstellung von bewegten Bildern zu ermöglichen, werden die Image-Bit-Planes des →Bildspeichers doppelt ausgelegt. Während der Inhalt des einen Bildspeichers angezeigt wird, wird in den anderen ein neues Bild geschrieben. Anschließend werden beide Speicher vertauscht. Dabei wird das neue Bild sofort sichtbar. Der störende Bildaufbau bleibt dem Betrachter hingegen verborgen.

Farbe stellt eines der wichtigsten gestalterischen Elemente bildlicher Darstellungen dar und ist auch in der wissenschaftlichen Visualisierung von zentraler Bedeutung, dort allerdings vorrangig als Träger darzustellender Informationen und erst in zweiter Linie als ästhetisch-gestalterisches Instrument.

Das menschliche Auge kann einen winzigen Ausschnitt des Spektrums elektromagnetischer Wellen als sichtbares Licht wahrnehmen (Wellenlänge $\lambda = 380-750nm$). Die menschliche Farbwahrnehmung hängt entscheidend ab von der Frequenzzusammensetzung der das Auge erreichenden Lichtwellen. Es gibt allerdings viele Faktoren, die die Farbwahrnehmung des Menschen – seine Reaktion auf ein sog. Farbereignis – unmittelbar oder mittelbar beeinflussen. Dazu zählen neben den physikalischen beispielsweise biochemische und psychologische Einflußgrößen.

Physikalisch kann ein Farbereignis durch eine spektrale Verteilungsfunktion repräsentiert werden. Sie beschreibt die Strahlungsenergiedichte als Funktion der Wellenlänge und definiert bei konstanten Einflußgrößen (s.o.) eindeutig eine Farbe, d.h. einen Farbeindruck. Diese Zuordnung ist allerdings nicht eineindeutig, unendlich viele spektrale Verteilungsfunktionen können den gleichen Farbeindruck hervorrufen (→Farbwahrnehmung). Unterschiedliche spektrale Verteilungen, die als gleichartig wahrgenommen werden, werden *Metamere* genannt. Es ist ein Ziel der →*Colorimetrie*, eine eineindeutige Beschreibung von Farben bereitzustellen.

Die Farbmodelle der Computergrafik sollen dem Benutzer durch wenige Parameter ermöglichen, eindeutig Farben aus einem möglichst großen Bereich darstellbarer Farben (*Gamut*) zu spezifizieren (→Farbmodelle). Das weithin verbreitete RGB-Modell z.B. entspricht aus dieser Sichtweise einer Approximation von spektralen Verteilungsfunktionen durch 3-Punkt-Sampling.

Die Verwendung von genau drei Parametern zur eindeutigen Charakterisierung einer Farbe basiert auf sinnespsychologischen Versuchen zum *Color-Matching* (→Farbwahrnehmung, Trichromizitätstheorie), aus denen hervorgeht, daß jeder wahrnehmbare Farbeindruck durch eine Linearkombination dreier linear unabhängiger Farben dargestellt werden kann – allerdings nur, wenn bis zu zwei Gewichtungen negativ sein dürfen, was z.B. im RGB-Modell nicht der Fall ist.

Eine ausführliche Darstellung des Themas Farbe aus wissenschaftlicher Sicht findet sich in [74], für eine Einführung in die psychologischen Farbwirkungen und deren Nutzung in „visuellen Formulierungen“ siehe etwa [4].

Farbmodell. In der Computergrafik werden Farben häufig durch *RGB*-Tripel beschrieben. Ein *RGB*-Tripel gibt an, wieviel Rot-, Grün- und Blauanteil in der Farbe vorhanden ist. Maximale Werte von Rot, Grün und Blau zusammen ergeben Weiß. Deshalb spricht man von einem additiven Farbsystem. Es verhält sich so, als würde man farbiges Licht addieren. Ein alternatives Farbsystem ist das *CMY*-Modell der Komplementärfarben (*CMY = cyan, magenta, yellow*). Dabei handelt es sich um ein subtraktives Farbsystem. Cyan, Magenta und Gelb zusammen ergeben bei subtraktiver Farbmischung Schwarz.

Unter den vielen anderen existierenden, zum Teil auch genormten Farbsystemen ist insbesondere noch das *HSV*- oder *HSB*-Modell — *H*ue (Farbwert), *S*aturation (Sättigung, Reinheit) und *V*alue resp. *B*rightness (Helligkeitswert) — erwähnenswert. Als leichte Abwandlung davon spielt auch das *HLS*-Modell eine Rolle. *HLS* steht für *H*ue, *L*ightness und *S*aturation.

Die drei Parameter *hue, saturation, value* stammen ursprünglich aus sinnespsychologischer Nomenklatur; ein solches Tripel war geeignet, eine subjektive vollständige Beschreibung eines Farbeindrucks vorzunehmen. Das *HSV*-Computer-Farbmodell bietet eine quantifizierte und normierte Beschreibung dieser Parameter; es ist in dieser Form isomorph zum RGB-Modell. Der Zusammenhang mit dem *RGB*-Modell ist wie folgt: Alle möglichen *RGB*-Werte spannen einen Würfel auf, in dem sich Schwarz und Weiß gegenüber liegen. Die drei Grundfarben sowie die Komplementärfarben bilden, projiziert auf eine Ebene senkrecht zur Schwarz-Weiß Diagonale, ein Sechseck. *Value* bezeichnet nun den Abstand dieser Ebene zum Ursprung. Je kleiner dieser Wert ist, desto dunkler ist die Farbe. *Saturation* gibt den Abstand des Farbpunktes von der Schwarz-Weiß-Diagonalen in der Projektionsebene an. Je größer dieser Wert ist, umso kräftiger gesättigt ist die Farbe. *Hue* gibt den Winkel des Farbpunk-

tes in der Projektionsebene relativ zu Rot an. Die möglichen Färbungen werden in der Reihenfolge Rot, Gelb, Grün, Cyan, Blau und Magenta durchlaufen. Der aufgespannte Farbraum hat die Form eines sechseckigen Kegels. Im Gegensatz dazu hat der Farbraum im *HLS*-Modell die Form eines sechseckigen Doppelkegels, wobei an der zusätzlichen Spitze Weiß angesiedelt ist.

Anspruchsvolle Renderer (wie z.B. der *Wavefront* Renderer) beschreiben Materialien und Lichtquellen unter Angabe ihrer Spektralkurven. Für den Rendervorgang kann dann vom Benutzer angegeben werden, wie viele *point samples* aus dem Spektralbereich verwendet werden sollen. Default ist allerdings auch hier drei, was einer Beschreibung mit dem RGB-Modell gleichkommt.

Es wird an Modellen gearbeitet, die jede spektrale Verteilungsfunktion (Farbe) einer Szene mit Hilfe festgelegter orthonomaler Basisfunktionen repräsentieren und somit grundsätzlich eine genauere Modellierung der Farbphänomene und Reflektionseigenschaften der in der Szene vorhandenen Objekte ermöglichen, ohne zu übermäßigem Mehraufwand bzgl. der Rechenkosten zu führen [49].

Farbtabelle. Ein Bildspeicher mit n →Bit-Ebenen erlaubt die Abspeicherung von 2^n verschiedenen Werten. Standard für →True-Color-Darstellungen ist heute $n = 3 * 8$, also $2^{24} \approx 16 * 10^6$ Farben insgesamt. Um mit weniger Bit-Ebenen auszukommen, speichert man im Bildspeicher oft nicht die Pixelwerte, sondern Indizes, die auf Einträge einer *Farbtabelle (colormap)* verweisen (pseudo color, indexed color). Die Farbtabelle kann für jedes Bild geändert werden. Die Zahl der Bit-Ebenen im Framebuffer limitiert nicht mehr die Zahl der insgesamt möglichen Farben, sondern nur noch die Zahl der *gleichzeitig*, also in einem Bild darstellbaren Farben. Typischerweise ist der Farbindex 8 oder 12 Bit lang; die Farbtabelle umfaßt entsprechend 256 oder 4096 Einträge. Jeder Eintrag ist meist 24 Bit lang, also auswählbar aus einer Palette von $16 * 10^6$ Farben.

Rasterbilder werden oft auf gleiche Weise in Dateien gespeichert. Einige Formate kennen nur Indizes in eine Standardfarbtabelle, die nicht explizit in der Bilddatei enthalten ist. Andere Formate erlauben beliebige Farbtabellen die natürlich in der Bilddatei mit abgespeichert werden müssen.

Enthält ein berechnetes Bild mehr Farben als die Farbtabelle lang sein darf, muß →Halftoning angewandt werden.

Farbwahrnehmung Für die Umwandlung von sichtbarem Licht in elektrische Neuro-Impulse sind zwei Arten von Rezeptorzellen in der Netzhaut (Retina) des menschlichen Auges verantwortlich: die sogenannten Stäbchen und Zäpfchen. In diesen Zellen sorgen spezielle Moleküle, die visuellen Pigmente, auf Lichteinfall hin für die Erzeugung der elektrischen Impulse.

Die Stäbchen sind hauptsächlich für das Sehen bei schwacher Beleuchtung zuständig und dienen der Unterscheidung von Intensitätsunterschieden

(Schwarz-Weiß-Sehen). Die Zäpfchen sind für die Farbwahrnehmung und für detailreiches Sehen bei starkem Lichteinfall zuständig.

In der menschlichen Netzhaut gibt es ca. 120 Millionen Stäbchen und 6 Millionen Zäpfchen. Diese sind jedoch nicht gleichmäßig über die Retina verteilt, in der *Fovea*, dem winzigen Punkt des schärfsten Sehens, sind z.B. keine Stäbchen, dafür eine sehr hohe Konzentration von Zäpfchen vorhanden. In der Umgebung der Fovea ist die Konzentration der Rezeptoren größer als zum Rand der Netzhaut hin.

Es gibt in der menschlichen Netzhaut drei verschiedene Zäpfchentypen mit Pigmenten, die auf Licht verschiedener Wellenlänge unterschiedlich reagieren. 1965 konnten die Antwortfunktionen für die drei Pigmente ermittelt werden. Die sich ergebenden Kurven für die Absorptionsspektra (Absorption über Wellenlänge) sind sich in der Form sehr ähnlich, haben aber Maxima bei 419 nm (blau), 531 nm und 558 nm (grün–gelb) respektive.

Dieser Befund war ein Beleg für die sog. *Trichromatische Hypothese*[4]. Diese Theorie besagt, daß das Farbensehen auf drei Rezeptormechanismen beruht, von denen jeder auf unterschiedliche spektrale Reize anspricht. Licht einer bestimmten Wellenlängenverteilung stimuliert die drei Mechanismen in unterschiedlichen Ausmaßen, und das Verhältnis der Stimulationen zueinander repräsentiert einen empfundenen Farbeindruck. Die Hypothese entstand als Reaktion auf die Ergebnisse von sogenannten *Color Matching*-Versuchen, in denen Beobachter ein vorgegebenes farbiges Licht (Testlicht) aus drei ebenfalls vorgegebenen Wellenlängen (Basislichtern) zusammenmischen sollte. Beobachter mit normaler Farbempfindung waren immer fähig, aus drei Farben das Testlicht zu mischen, wenn sie beliebige Gewichtungen vornehmen durften, von denen auch bis zu zwei negativ sein durften, d.h., ein Licht vom entsprechenden Betrag wurde, statt vom Vergleichsfeld abgezogen, zum Testlicht hinzuaddiert. Zwei Farben reichen nicht aus, um ein beliebiges Testlicht zusammenzumischen.

Eine weitere frühe Theorie zur Farbwahrnehmung, die sog. *Komplementärfarbenhypothese* (Opponent Process Hypothese)[5], ist ebenfalls durch neurologische Befunde unterstützt worden und ergänzt das Modell des Farbensehens. Man geht heute von einer Mehr-Stufen-Theorie (*zone theory*) der menschlichen Farbwahrnehmung aus, nach der in der ersten Stufe die drei Zäpfchentypen elektrische Impulse entsprechend ihrer Absorption erzeugen, und nachgeschaltet Nervenzellen auf diesen Impulsen eine Verarbeitung nach der Komplementärfarbentheorie vornehmen, wodurch nah beeinanderliegende Farbreize verstärkt werden und insgesamt ein einfacheres Signal in tiefere Hirnregionen weitergeleitet wird.

Bei fester Intensität und Sättigung und variabler Wellenlänge können Betrachter im sichtbaren Spektralbereich (380-700 nm) circa 150 verschiedene Farbwerte unterscheiden (experimentell). Man hat unter Berücksichti-

[4]Thomas Young 1802, Hermann v. Helmholtz 1852
[5]Edward Hering, 1878

gung weiterer experimenteller Befunde hochgerechnet, daß vom Menschen bis zu 7 Millionen Farben unterschieden werden können.

Flat-Shading, siehe →Schattierung.

Frame-Buffer, siehe →Bildspeicher.

Gamma-Korrektur. Bei Monitoren ist der Zusammenhang zwischen der Spannung U an der Bildröhre und der resultierenden Intensität I nicht linear, sondern gehorcht einem Potenzgesetz, $I = \text{const.}\, U^{\gamma}$. Der Wert für γ liegt meistens im Bereich $1 \leq \gamma \leq 4$. Um unabhängig vom verwendeten Monitor die gleiche Intensität zu erzielen, müssen die Pixelwerte eines Rasterbildes entsprechend dem γ-Wert des jeweiligen Monitors korrigiert werden (*gamma correction*). Diese Korrektur wird jedoch in der Regel von der Grafikhardware vorgenommen. Der voreingestellte Gamma-Wert ist oft durch Software veränderbar. Neben der Intensität ist natürlich auch die Farbe selbst vom verwendeten Monitor abhängig. Obwohl die meisten Monitore sich an einem Farbstandard orientieren, gibt es starke Abweichungen zwischen den verschiedenen Fabrikaten, wie auch fertigungsbedingte Schwankungen. Auf Korrekturen wird jedoch meist verzichtet. Das Thema „Farbtreue“ ist zu komplex, um es hier zu skizzieren (s. [27, 10]).

Gouraud-Shading, siehe →Schattierung.

Halftoning. Unter *halftoning* versteht man Techniken, ein beliebiges Farb- oder Graustufenbild mit möglichst geringen Verlusten auf ein Bild zu reduzieren, das nur einen beschränkten Satz von Farben enthält. Ein typisches Beispiel sind Bilder in einer Tageszeitung, wo Graustufen durch ein Muster verschieden großer schwarzer Punkte dargestellt werden. Oft wird nur eine einfache Farbquantisierung (*color quantisation*) durchgeführt. Dabei wird eine nicht darstellbare Farbe durch die am nächsten gelegene darstellbare Farbe ersetzt. Im Extremfall wird aus jedem dunklen →Pixel ein schwarzes, aus jedem hellen ein weißes. Deutlich bessere Ergebnisse erzielt man durch Nachschalten des Algorithmus von *Floyd* und *Steinberg*, der Fehler bei Quantisierung auf benachbarte Pixel verteilt.

Durch sogenanntes *Dithering* kann man die Zahl der möglichen Grau- oder Farbstufen auf Kosten der räumlichen Auflösung erhöhen, indem man Grau- bzw. Farbwerte durch geeignet gemusterte Rechtecke von mehreren Rasterpunkten emuliert.

HLS, siehe →Farbmodell.

HSV, siehe →Farbmodell.

Illumination Model. Siehe →Beleuchtungsmodell.

Interleave-Method. Bei RGB-kodierten Bildern bezeichnet die Interleave--Methode die Art und Weise, wie die einzelnen Farbwerte in einer Bilddatei

aufeinander folgen. Üblich sind die Angabe vollständiger RGB-Tripel hintereinander (RGB RGB RGB.., *noninterleaved*), die zeilenweise Aufeinanderfolge (RR..GG.. BB.. RR..GG..BB.., *scanline-interleaved*) und die bildweise Aufeinanderfolge (RRR..BBB..GGG, *plane-interleaved*). Einige Bildformate unterstützen alle drei Methoden, andere nur zwei oder eine einzige.

Kanal. Die Anzahl der Kanäle (Channels) in einem Bild ist die Anzahl der gespeicherten Werte pro →Pixel. In einem Grauwertebild oder in einem Bild mit Indizes auf eine →Farbtabelle gibt es nur einen Kanal. In einem RGB-kodierten Bild (s. →RGB) sind es drei Kanäle, je einer für rot, grün und blau. Ein zusätzlicher Kanal entsteht durch Hinzunahme von Alphawerten, die die Transparenz angeben (→Alpha Channel). Als *Channel Depth* wird die Anzahl der Bits pro Kanal bezeichnet.

Klippen, (engl. Clipping). Das Entfernen der Teile von Elementen der Seiten- bzw. Szenenbeschreibung, die außerhalb einer spezifizierten Fläche (Klipprechteck) bzw. Volumens (Klippvolumen) liegen.

Lauflängen-Kodierung, engl. Run-Length Encoding, ist ein verbreitetes Verfahren zur platzsparenden Ablage von Rasterbildern (→Bildkompression). Es wird ausgenutzt, daß in den Zeilen eines Rasterbildes aufeinanderfolgende →Pixel oft die gleiche Farbe haben, wie das zum Beispiel bei einem einfarbigen Hintergrund der Fall ist. Solche Folgen gleichfarbiger Pixel lassen sich zweckmäßig durch die Folgenlänge und eine einmalige Angabe des Farbwertes beschreiben. In der Regel erreicht man durch solch eine Kodierung des Bildes eine Reduktion auf 50-70% der ursprünglichen Größe. Bei einfachen Bildern ohne ausgedehnte und stetige Farbverläufe kann der Reduktionsfaktor auch deutlich günstiger sein.

Lighting Model, siehe →Beleuchtungsmodel.

Liniengrafik, auch *Vektorgrafik* genannt, bezeichnet die Bildberechnung und Darstellung auf der Basis von Linien im Gegensatz zu →*Rastergrafik*. Auf Vektorbildschirmen werden Linienzüge, Text- und Grafiksymbole durch Linien gleicher Helligkeit und mit exakten Anfangs- und Endpunkten *analog* erzeugt. Liniengrafik wurde in den letzten Jahren fast vollständig durch →Rastergrafik abgelöst; nur in wenigen Anwendungsgebieten konnte sich die Liniengrafik noch halten.

Magic Number. Viele Formate fordern zur Identifikation eine bestimmte Zeichen- oder Bytekombination (meist 2 Bytes) am Beginn eines Files (etwa `%!` bei PostScript). Diese Kennung wird *magic number* genannt.

Phong-Shading, siehe →Schattierung.

Pixel. Einzelner Punkt in einem Bildraster (s. →Rastergrafik).

Pseudo-Color, siehe →Farbtabelle.

Radiosity. Verfahren zur Berechnung von realistischen Bildern. Hierbei wird für jede Fläche einer 3-D-Szene die Lichtenergie berechnet, die diffus (also gleichmäßig in alle Richtungen) von der Fläche abgestrahlt wird. Diese Energiemenge hängt von der Lichtenergie ab, die eine Fläche aus sich heraus abstrahlt und die sie von anderen Flächen empfängt und nicht absorbiert. Nach Bestimmung der wechselseitigen Verdeckungsverhältnisse aller Flächenpaare im Objektraum wird durch Lösung eines Integralgleichungssystems das *Strahlungsgleichgewicht* der Szene berechnet. Diese Berechnung ist, im Gegensatz zum →*Raytracing*, unabhängig von der Sicht auf die Szene. Nach diesem aufwendigen Preprocessing kann die Bildberechnung für beliebige Blickpunkte schnell geschehen: sie erfordert nur noch Hidden-Surface-Removal, etwa durch Tiefensortierung. Die Stärke des Radiosity-Verfahrens ist die genaue Modellierung der diffusen Reflexion zwischen Objekten, da sie zu recht realistisch aussehenden Bildern führt.

Zur Synthese (fast) fotorealistischer Bilder kann das Radiosity-Verfahren so verallgemeinert werden, daß auch komplexere, winkelabhängige Abstrahlcharakteristiken, bis hin zu spekulären Reflexionen berücksichtigt werden (u.a. durch Kombination mit Raytracing-Verfahren).

Rastergrafik. Bildberechnung und -darstellung auf der Basis von →Pixeln, im Gegensatz zu →*Liniengrafik*. Geeignete Hardware-Unterstützung verhalf der Rastergrafik in den letzten Jahren trotz anfänglichen Speicher- und CPU-Engpässen zum Durchbruch. Inzwischen hat sie die Liniengrafik fast vollständig abgelöst.

Raytracing, dt. Strahlverfolgungsverfahren, ist ein Verfahren zur Berechnung von annähernd fotorealistischen Bildern von 3-D-Szenen. Für jeden Punkt der Bildfläche wird (mindestens) ein Sehstrahl in die darzustellende Szene geschickt (*backward raytracing*). Bei jedem Auftreffen auf ein anderes Medium wird der Strahl entsprechend den Eigenschaften des Mediums reflektiert und/oder gebrochen und transmittiert. Dabei wird der Farbwert des Strahls durch das jeweilige Medium verändert. Die Strahlen werden rückwärts verfolgt, bis sie eine Lichtquelle treffen, ins Unendliche gehen oder bis die Zahl der Reflektionen und Brechungen eine gewisse vorgebene Grenze überschreitet. Diese Simulation der Intensitäts- und Farbverhältnisse von 3-D-Szenen ergibt halbwegs realistische Bilder. Insbesondere Spiegelungen werden recht naturgetreu wiedergegeben.

Ein anderes Verfahren zur Erzeugung realistisch wirkender Bilder ist das →*Radiosity-Verfahren*, in welchem das Strahlungsgleichgewicht durch die wechselseitige, diffuse Bestrahlung der Objekte berechnet wird. Dem Fotorealismus am nächsten kommt man durch geeignete Kombination dieser beiden Verfahren.

Rendering bezeichnet die Bildberechnung, d.h. die Erzeugung der Ausgabeprimitive, also →Pixel oder Linien, aus denen sich das Bild zusammensetzt

(s. →Rastergrafik und →Liniengrafik). Heute, in der Zeit der Rastergrafik, ist mit Rendering fast immer die Berechnung von Rasterbildern gemeint.

Die Bildberechnung geschieht auf Basis einer vorgegebenen Seiten- bzw. Szenen-Beschreibung. Diese enthält Form, Position, Oberflächen- und Volumeneigenschaften der 2-D- bzw. 3-D-Objekte, den Blickpunkt und die Projektionsart bzw. die Eigenschaften der (virtuellen) Kamera. Bei Szenen der Dimension $D > 2$ tritt zur Unterstützung des räumlichen Eindrucks die Berücksichtigung von Lichtquellen hinzu. Deren Position, geometrische und optische Strahlungscharakteristiken sind auch Inhalt der Szenenbeschreibung. Wichtigste Teilaufgaben der Bildberechnung sind die Berechnung der sichtbaren Flächen, der Farb- und Helligkeitswerte auf diesen sowie deren Abbildung auf die Punkte des Bildrasters. Zur visuellen Datenanalyse werden auch andere Verfahren angewandt, etwa solche, die aus vorgegebenen Feldwerten auf 2- oder 3-D-Mannigfaltigkeiten des R^n durch Einfärbung bzw. direktes →Volume-Rendering Bilder berechnen.

Je nach Aufgabenstellung, Qualitäts- und Geschwindigkeitsanforderungen kommen beim Rendering eine Reihe unterschiedlicher Teilalgorithmen zur Anwendung. Typische Algorithmen zur *Verdeckungsberechnung* sind Z-Buffer-, Scan-Line-, Prioritätslisten- und rekursive Unterteilungsalgorithmen. Bei der *Farbwertberechnung* werden unterschiedliche Interpolationsmethoden (→Schattierung) und →Beleuchtungsmodelle verwendet. Da im Raum plazierte Objekte sich in der Realität gegenseitig bestrahlen, sind für fotorealistische Bildberechungen nicht nur die sich dem Beobachter präsentierenden Verdeckungsverhältnisse relevant. Aufwendige Rendering-Verfahren, wie →Ray-Tracing- und →Radiosity-Verfahren berücksichtigen daher die wechselseitigen Verdeckungs- und Bestrahlungsverhältnisse aller in der Szene plazierten Objekte.

Run-Length Encoding, siehe →Lauflängenkodierung.

RGB. Um die Farbe eines Bildpunktes zu beschreiben, wird häufig ein *RGB*-Tripel verwendet. Die Komponenten geben die Anteile von rotem, grünem und blauem Licht an. Oft werden ganze Zahlen von 0 bis 255 oder Dezimalbrüche zwischen 0 und 1 benutzt. Dann bedeutet (0,0,0) schwarz und (255,255,255) bzw. (1,1,1) weiß. Weiteres unter →Farbmodell.

Schattierung, engl. *shading*, ist die Erzeugung geeigneter Farbverläufe in der Abbildung einer 3-D-Szene, insbesondere solchen, die den räumlichen Sichteindruck unterstützen. Shading ist somit ein Teil der Bildberechnung (→Rendering).

Intensität und Farbe auf Punkten eines Flächenelementes sind durch dessen Position, Neigung und Oberflächeneigenschaften, durch den Standpunkt der virtuellen Kamera und durch das →Beleuchtungsmodell gegeben. Zur Rechenersparnis werden die von so vielen Parametern abhängigen Werte oft nicht für jeden Punkt eigens berechnet, sondern nur für Eckpunkte (ebener) Flächenelemente. Die übrigen Werte werden durch

Interpolation auf verschiedenen Ebenen gewonnen. Man unterscheidet zwischen dem *konstanten Schattieren (flat shading)* – jeder Punkt des Flächenelementes erhält die gleiche Farbe – dem *Gouraud-Shading* – lineare Interpolation der Farben über der Fläche – und dem *Phong-Shading* – Interpolation entlang der Normalen über der Fläche und der sich hiermit ergebenden Farben gemäß dem verwendeten →Beleuchtungsmodell.

In älteren Grafiksystemen (X, GKS, PHIGS) sind oft spezielle Ausfüllungsarten von Flächen vorgesehen, etwa Schraffuren (hatch style). Heute realisiert man diese Effekte durch das allgemeinere Konzept der →Textur-Abbildung.

Shading, siehe →Schattierung.

Shading Model, siehe →Beleuchtungsmodell, →Schattierung.

Textur. Die Oberflächenstruktur, etwa Rauhigkeit oder farbliche Musterung, eines Körpers.

Textur-Abbildung. Ein allgemeines Hilfsmittel, um lokale Variationen von Oberflächeneigenschaften zu definieren. Urbildbereich einer *Textur-Abbildung* ist die Objektoberfläche, deren Eigenschaften durch ortsabhängige Parameter zu beschreiben sind. Bildbereich der Abbildung ist die *Textur*, ein 1,2 oder 3-dimensionales Feld von Daten, welche die möglichen Parameterwerte darstellen. Eine einfache und häufige Wahl für das *Texturvolumen* ist ein vorberechnetes Rasterbild. Jedem Punkt der Objektoberfläche werden Texturkoordinaten, d.h. ein Punkt (texture vertex) im Texturvolumen zugeordnet. Die Textur-Abbildung wird häufig dadurch definiert, daß man den Eckpunkten der polygonal definierten Oberfläche Texturkoordinaten fest zuordnet und solche für die anderen Oberflächenpunkte hieraus durch Interpolation berechnet. Aus den Texturkoordinaten ergibt sich durch ein Table Lookup der entsprechende Texturwert. Bei sogenannten *prozeduralen Texturen* werden die Werte im Texturvolumen nicht in Form eines Datenfeldes vorgegeben, sondern prozedural berechnet – oft unter Nutzung von Zufallszahlen. Typische Parameter, deren Werte durch Texturabbildungen definiert werden, sind Farbe, Störung der Flächennormalen (sog. *bump mapping*), Reflektion und Transparenz. Textur-Abbildungen erlauben es auf einfache Weise, die abzubildende Szene mit visuellen Details und Komplexität zu versehen, so daß ein subjektiv realistisches Erscheinungsbild resultiert.

True Color. Die Farbe eines →Pixels wird meist als RGB-Wert (s. →Farbmodell, →RGB) angegeben, mit mindestens 8-Bit Farbtiefe je Komponente. Daraus ergeben sich mehr als $16 * 10^6$ Farben, die *gleichzeitig* dargestellt werden können. Wird dagegen eine →Farbtabelle (*Colortable*) verwendet, um den Farbwert eines Pixels anzugeben, ist die Anzahl der gleichzeitig darstellbaren Farben meist eingeschränkt und man spricht nicht mehr von *True Color*.

Volume-Rendering. Im weiteren Sinne Sammelbegriff für verschiedene *Rendering-Verfahren* (→Rendering) zur Visualisierung höherdimensionaler ($D \geq 3$) skalarer, vektorieller und tensorieller Felder. Im engeren Sinne versteht man unter Volume-Rendering oft Verfahren zur Visualisierung des einfachsten und in Anwendungen häufigsten Falles, nämlich 3-dimensionaler, skalarer Felder. Im folgenden wird nur dieser Fall betrachtet.

Man unterscheidet zwei Typen von Methoden, nämlich solche, die *Isoflächen* berechnen (polygonale oder andere) und daraus mit Standardmethoden Bilder erzeugen und solche, die Daten direkt als mehr oder weniger transparente Dichtewolken darstellen (*direct volume rendering*).

Im ersten Fall tragen nur diejenigen Daten zum Bild bei, welche die Isoflächen definieren. Dabei können jedoch durchaus mehrere ineinanderliegende Flächen dargestellt werden, wenn man diese transparent macht. Im zweiten Fall kann dagegen jedes Volumenelement das endgültige Bild beeinflussen.

Beim direkten Volume-Rendering unterscheidet man zwischen *Ray-Casting-Methoden* und *Projektions-Methoden*. In beiden Fällen werden für alle Volumenelemente die emitierten, absorbierten und gestreuten Intensitätsbeiträge berechnet. Aus diesen Beiträgen läßt sich das fertige Bild konstruieren. In den meisten Fällen werden Streueffekte höherer Ordnung vernachlässigt. Dies führt dazu, daß die Intensitätsbeiträge der einzelnen Volumenelemente nur jeweils entlang einer geraden Linie, dem Sehstrahl, aufintegriert werden müssen. Beim Ray-Casting werden Pixel für Pixel Sehstrahlen in das Volumen hineingesendet und unmittelbar integriert. Bei den Projektionsmethoden werden dagegen umgekehrt alle Volumenelemente in geordneter Weise durchlaufen und in die Bildebene projiziert. Für jedes Pixel werden Farbe und Transparenz entsprechend aktualisiert. Hierbei können auch Möglichkeiten moderner Grafik-Hardware genutzt werden.

Voxel. Ein quaderförmiges ($D \geq 3$) Volumen-Element. Im Unterschied zum →Pixel (rechteckiges 2-D-Element) ist ein Voxel nicht Element des Bildraumes, sondern des darzustellenden Objektraumes.

Z-Buffer. Um Flächen auf Verdeckung abzuprüfen, wird bei der Anwendung des *Z-Buffer-Verfahrens* zusätzlich zu dem Farbwert jedes →Pixels auch die Z-Koordinate des aktuellen Oberflächenpunkts im →Bildspeicher abgelegt (das Koordinatensystem sei so positioniert, daß die Z-Achse senkrecht aus dem Bildschirm ragt und $z = 0$ der hinteren Ebene des Objektraumes, genauer, des Klipp-Volumens entspricht). Der die Z-Werte fassende Teil des Bildspeichers heißt *Z-Buffer*. Soll an der gleichen Pixelposition ein neuer Farbwert (und damit auch Z-Wert) eingetragen werden, geschieht dies nur, wenn der schon vorhandene Wert im Z-Buffer für dieses Pixel kleiner ist als der neue Wert.

Anhang C

FAQ

Wie bekomme ich ein Farbbild vom Bildschirm einer Sun-Workstation auf Papier oder Folie? Mit dem OpenWindows-Tool *snapshot* (siehe Kap. 4.1.1) kann man den Inhalt eines Fensters, eines Teils des Bildschirms oder des gesamten Bildschirms in eine Datei sichern. Klickt man im Snapshot-Fenster "Print" und dort das Untermenü "Options..." an, erhält man ein weiteres Fenster, von dem aus man den Drucker (für Farbausgaben: cp) auswählen und für Farbausgaben die sog. Check Box "Monochrome Printer" ausschalten kann. Nach Anklicken des Knopfes "Print" wird der gewählte Teil des Bildschirmes gedruckt. Durch Auswahl von "Destination: File" wird nach Eingabe eines Dateinamens und Betätigen des Knopfes "Print" eine PostScript-Datei mit dem Bildschirminhalt erzeugt.

Wie integriere ich das ZIB-Logo in ein Bild, das auf Dia, Papier oder Folie ausgegeben werden soll? Das ZIB-Logo liegt als PostScript-Datei vor (`serv04:/zib/etc/dia/dia-ziblogo.ps`) und kann in *Framemaker* oder *Showcase* importiert werden. Eine weitere Möglichkeit besteht bei Bildern, die im PostScript-Format vorliegen: mit Hilfe des Kommandos *dia* kann das ZIB-Logo in die rechte obere Ecke des Bildes integriert werden (siehe Kap. 4.3, Seite 84). Beispielsweise zeigt

```
% dia -z -sc -l bild.ps
```

das Bild *bild.ps* mit ZIB-Logo auf dem Bildschirm an und druckt es auf dem Drucker *lw* aus.

Warum sollte man Screendumps nicht auf den Dia-Belichter geben? Der Dia-Belichter arbeitet mit einer Auflösung von 4096 x 2731 Pixel, was bei Projektion auf eine große Leinwand *gerade* ausreicht. Screendumps von typischerweise mehreren Hundert Pixeln Seitenlänge erscheinen bei Projektion auf eine Leinwand als grobes Raster; je größer die Leinwand, umso stärker der Effekt.

Weshalb Betacam? Reicht VHS oder S-VHS für eine Video-Aufzeichnung nicht aus? Video ist heutzutage noch meist ein analoges Medium. Die endlichen Bandbreiten begrenzen Farbauflösung und räumliche Auflösung. Ins-

besondere computer-generierte Bilder weisen Charakteristiken auf (etwa scharfe Kanten oder hohe Kontraste), die höchste Anforderungen an die Aufzeichnungstechnik stellen.

Auf Analog-Videoband werden unterschiedliche Informationsanteile (z.B. Farb- und Helligkeitsinformation) gemischt. Dies kann zu unangenehmen Effekten, z.B. 'auslaufenden' Farben, führen. Bei einfachen Aufzeichnungsformaten sind solche Probleme besonders ausgeprägt. Erfahrungen zeigen, daß in der wissenschaftlichen Visualisierung Betacam oder M-II das unterste noch akzeptable Qualitätsniveau darstellen. Besser geeignet wäre natürlich digitales Video.

Ein weiteres Argument gegen die Verwendung von VHS oder S-VHS sind die hohen Generationsverluste während des meist mehrstufigen Herstellungsprozesses. Man sollte versuchen, möglichst durchgängig mit hochwertigen Aufzeichnungsformaten zu arbeiten, und erst ganz am Schluß die Verteilkopien auf einfachere Formaten zu erzeugen. Falls man überhaupt eine Wahl hat, sollte man während des Herstellungsprozesses versuchen, das bestmögliche Format zu verwenden. Die heute gebräuchlichen Videoformate (in abnehmender Qualität) sind: D1, D2, MII, Betacam, Umatic SP, Umatic-Highband, S-VHS, Hi8, Umatic-Lowband und VHS. Im ZIB können wir derzeit auf Betacam, S-VHS und VHS aufzeichnen.

Wie produziere ich ein Real-Time-Video, obwohl mein Programm doch so langsam die Bilder berechnet? Nur in seltenen Fällen geschieht die Video-Aufzeichnung 'on-line', also parallel zur Bildberechnung. Normalerweise werden die Bilder im Voraus berechnet (pro Filmsekunde 25 Bilder für PAL bzw. 30 für NTSC) und in Form von Rasterfiles gespeichert. Dieser Vorgang darf beliebig lange dauern; erst wenn die Bildfolge fertig berechnet ist, erfolgt die Videoaufzeichnung (siehe Kap. 4.4).

Was ist PAL, Secam und NTSC? PAL, Secam und NTSC sind Fernsehnormen. Sie legen eine Reihe von Parametern (über 30) und Details fest, u.a. wieviele Bilder pro Sekunde dargestellt, wieviele Zeilen das Fernsehbild enthält, wie der Bildaufbau geschieht, welche Ablenkfrequenzen verwendet werden, usw. Diese Normen sind zu unterscheiden von den Aufzeichnungsformaten (VHS, S-VHS, Betacam usw.), die festlegen, in welcher Weise die Videosignale auf Band gespeichert werden.

Brauche ich für Vorfürungen im Ausland andere Videobänder? Im allgemeinen ja. Oft stehen nur Videoplayer zur Verfügung, die der landestypischen Fernsehnorm entsprechen, also etwa Secam in Frankreich oder NTSC in USA und Japan. Bei größeren, gut organisierten Konferenzen kann man aber zunehmend davon ausgehen, daß, unabhängig von Konferenzort auch PAL-Bänder abspielbar sind. Auf jeden Fall bei den Organisatoren rechtzeitig erkundigen und das Vorgehen mit der Abt. VisPar frühzeitig besprechen!

Können im ZIB Videos in NTSC-Norm erstellt werden? Nein. Derzeit stehen im ZIB keine Geräte zum Aufzeichnen von NTSC-Bändern zur Verfügung: somit können wir auch keine Videos von PAL nach NTSC kopieren.

Es besteht die Möglichkeit, bei einem professionellen Videostudio gegen entsprechendes Entgelt Videobänder von PAL nach NTSC und umgekehrt zu kopieren.

Welche Videomedien können wir im ZIB abspielen? Im Kursraum besteht die Möglichkeit, Videobänder mit Stereo-Ton und Display auf Fernsehmonitor oder Leinwand abzuspielen. Mögliche Formate sind Betacam (in PAL-Fernsehnorm) und S-VHS sowie VHS (in PAL, NTSC und Secam). Im Raum 407 können Videobänder mit Stereoton und unvertonte Bildplatten (Sony Analog Draw) mit Display auf Studiomonitoren abgespielt werden. Mögliche Formate sind Betacam (PAL) und S-VHS sowie VHS (PAL, NTSC und Secam) für Videobänder und Analog-Video-RGB für die Bildplatte.

Wie erhalte ich eine Bounding Box für eine PostScript-Datei? Die PostScript-Datei mit dem Programm *ghostview*, das auf allen Sun's und SGI-Workstations zur Verfügung steht, am Bildschirm darstellen! In der linken, oberen Ecke des Ghostview-Fensters sieht man dann Zahlen, die sich verändern, wenn man mit dem Maus-Cursor im Ghostview-Fenster herumfährt. Diese Zahlen sind die PostScript-Koordinaten des Punktes der aktuellen Cursorposition. Die Werte für die Bounding Box, welche in EPS-Files im Format
`%%BoundingBox: llx lly urx ury`
anzugeben sind, erhält man, indem man die Koordinaten der linken unteren (llx,lly) und der rechten oberen (urx,ury) Bildecke abliest. Diese Werte trägt man als zweite Zeile nach (%!...) in die PostScript-Datei ein. (s. Kap. 3.1.3)

Anhang D

Farbbilder

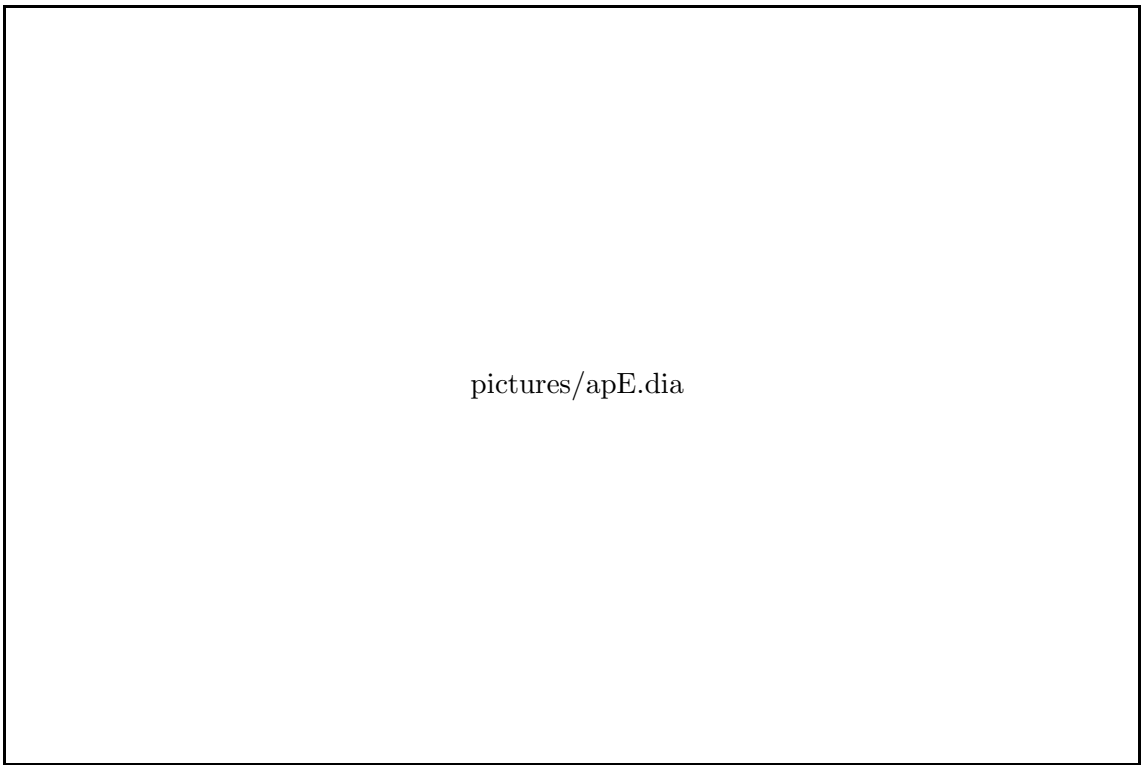

Abbildung D.1: Mit *apE* erzeugte Darstellung einer Tragflächenumströmung.

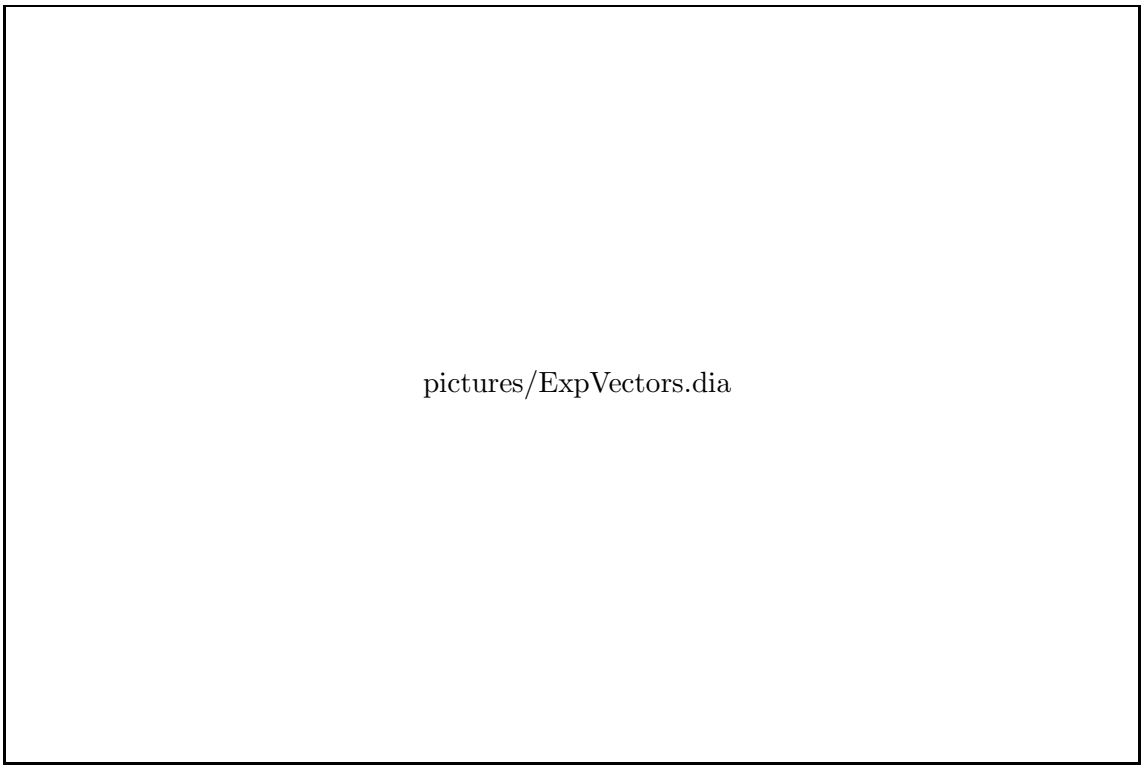

Abbildung D.2: Darstellung einer Strömung in einem Würfel (mit *IRIS Explorer*).

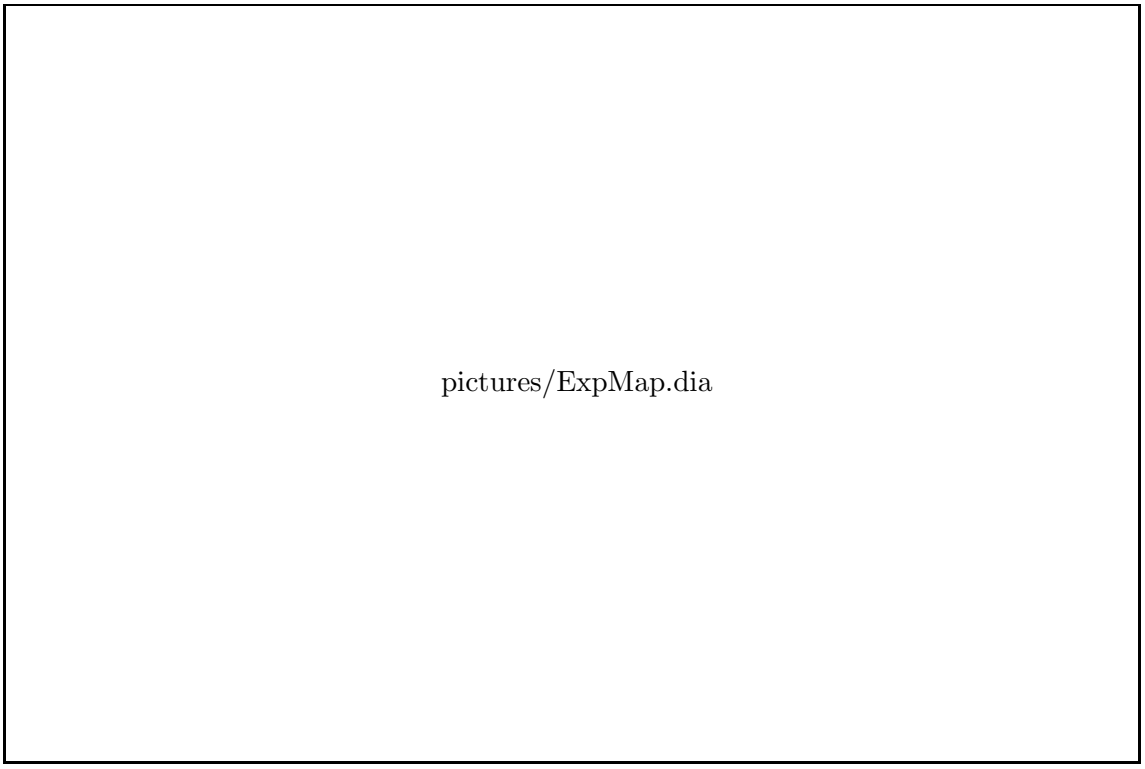

Abbildung D.3: Der Map Editor des *IRIS Explorer*.

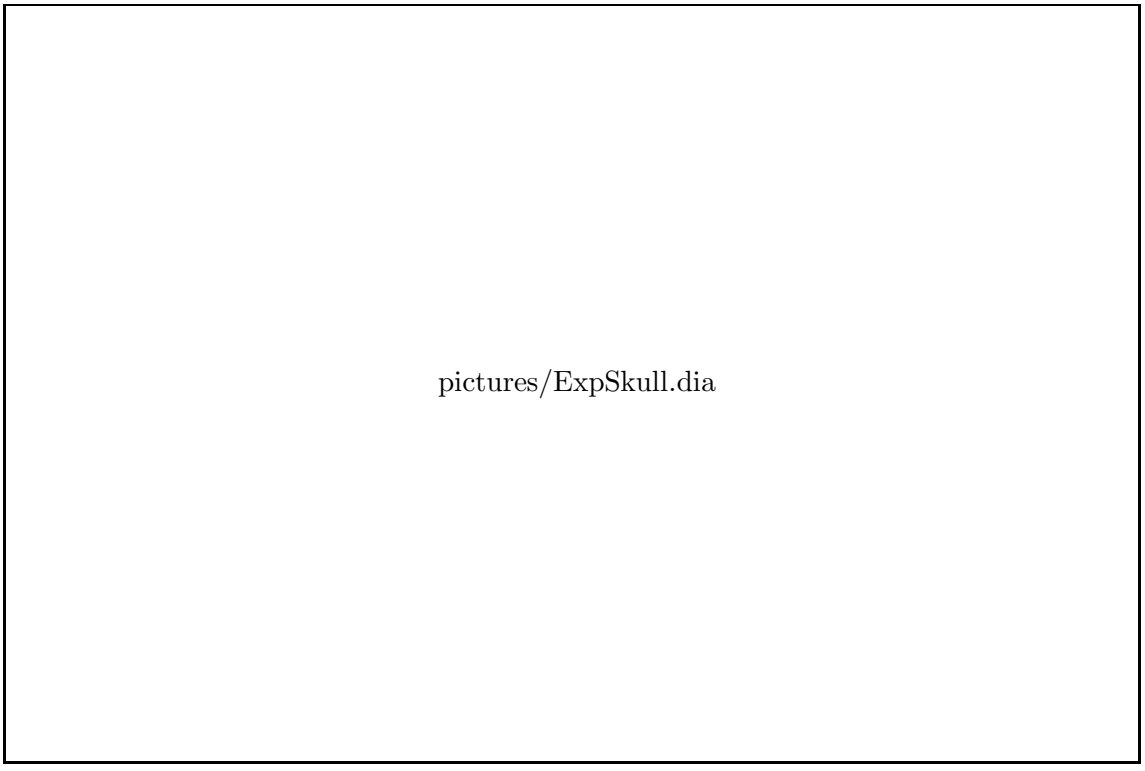

Abbildung D.4: Objekt, erzeugt mit der Map aus Abb. D.3.

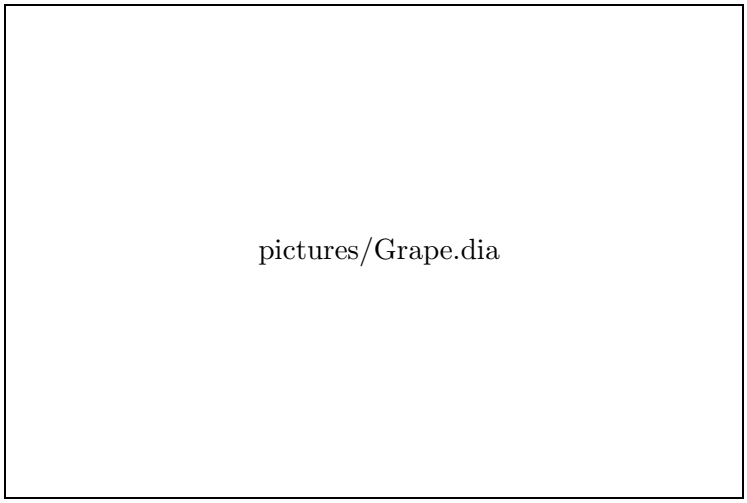

Abbildung D.5: Darstellungs- und Kontrollfenster von *GRAPE*.

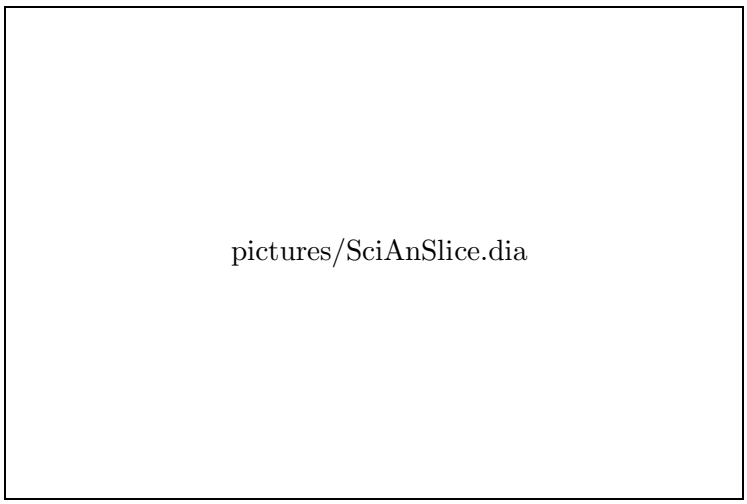

Abbildung D.6: Mit *SciAn* erzeugter 2-D-Schnitt durch ein Volumen.

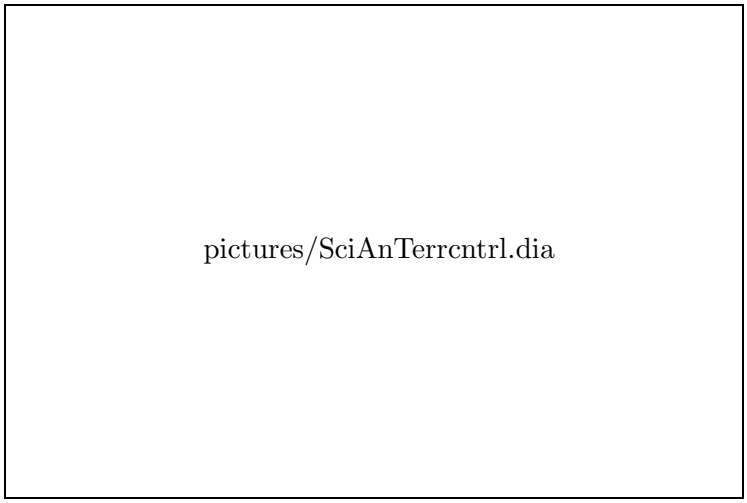

Abbildung D.7: Mit *SciAn* dargestelltes Volumen über einem Terrain.

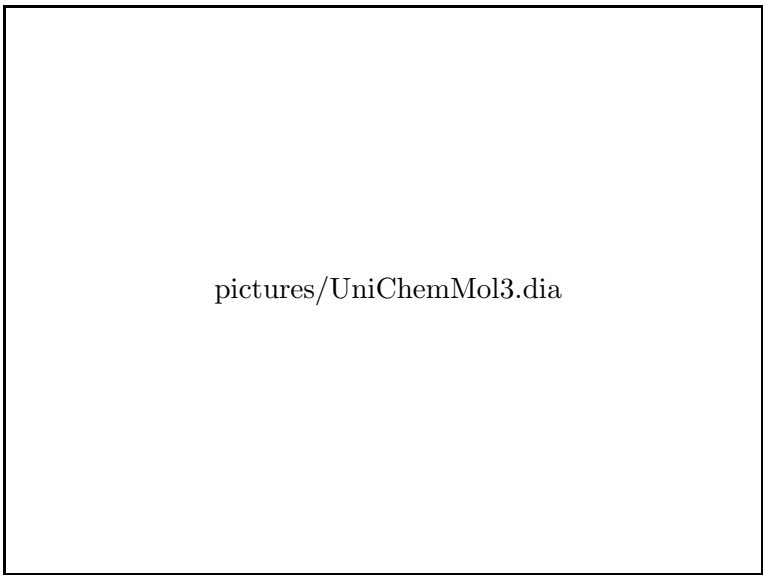

Abbildung D.8: Molekül, dargestellt mit *Rayshade* (Geometriedaten aus *UniChem*).

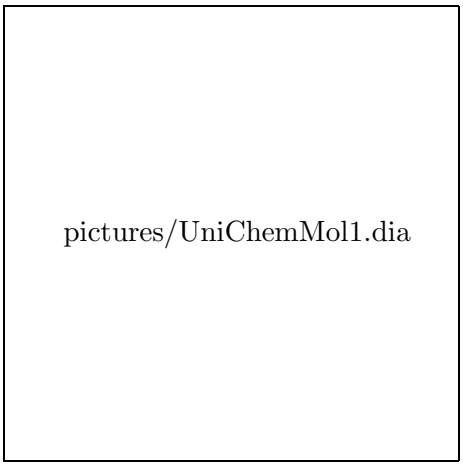

Abbildung D.9: Molekül, dargestellt mit *UniChem* (PostScript--Ausgabe).

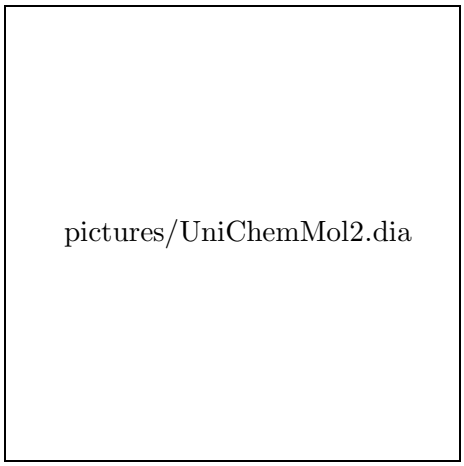

Abbildung D.10: Molekül, dargestellt mit *UniChem* an einer SGI-Workstation (Rasterbild).

Abbildung D.11: Molekül, dargestellt mit *AGIL* (auf der Basis von SunPHIGS).

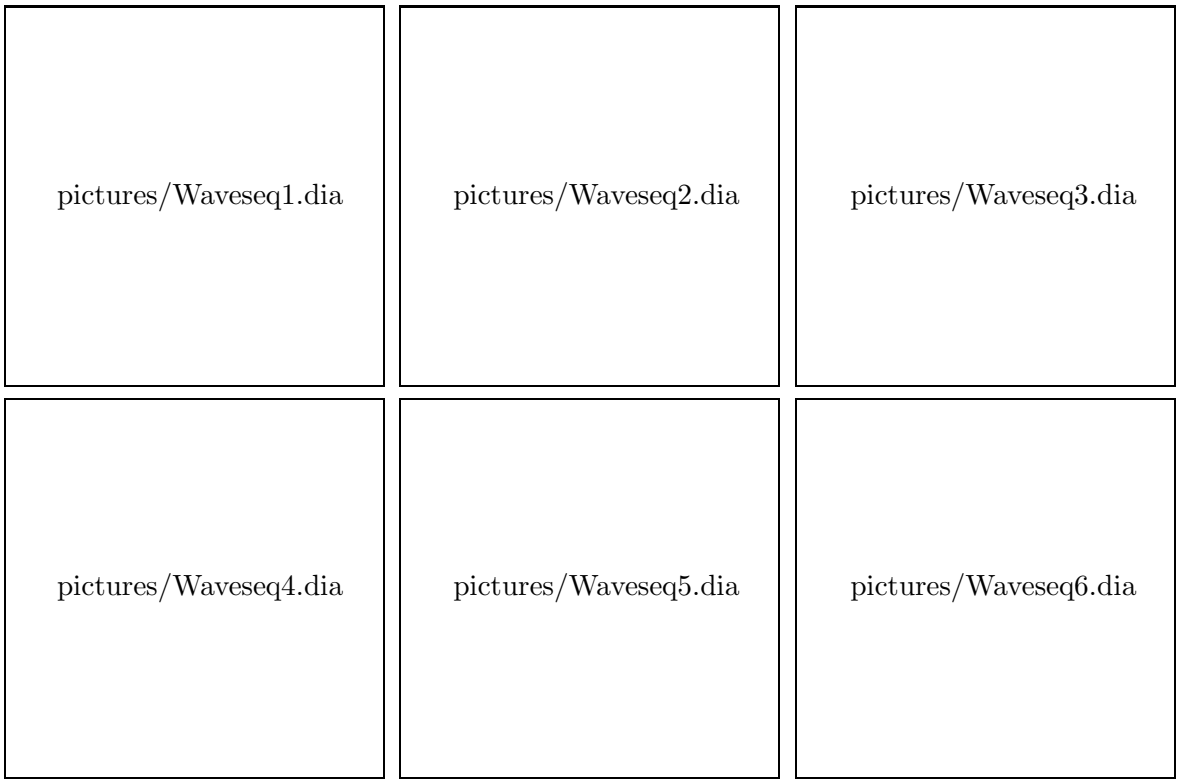

Abbildung D.12: Sequenz aus dem Vorspann des ZIB-Films, erstellt mit dem *Wavefront Advanced Visualizer*.

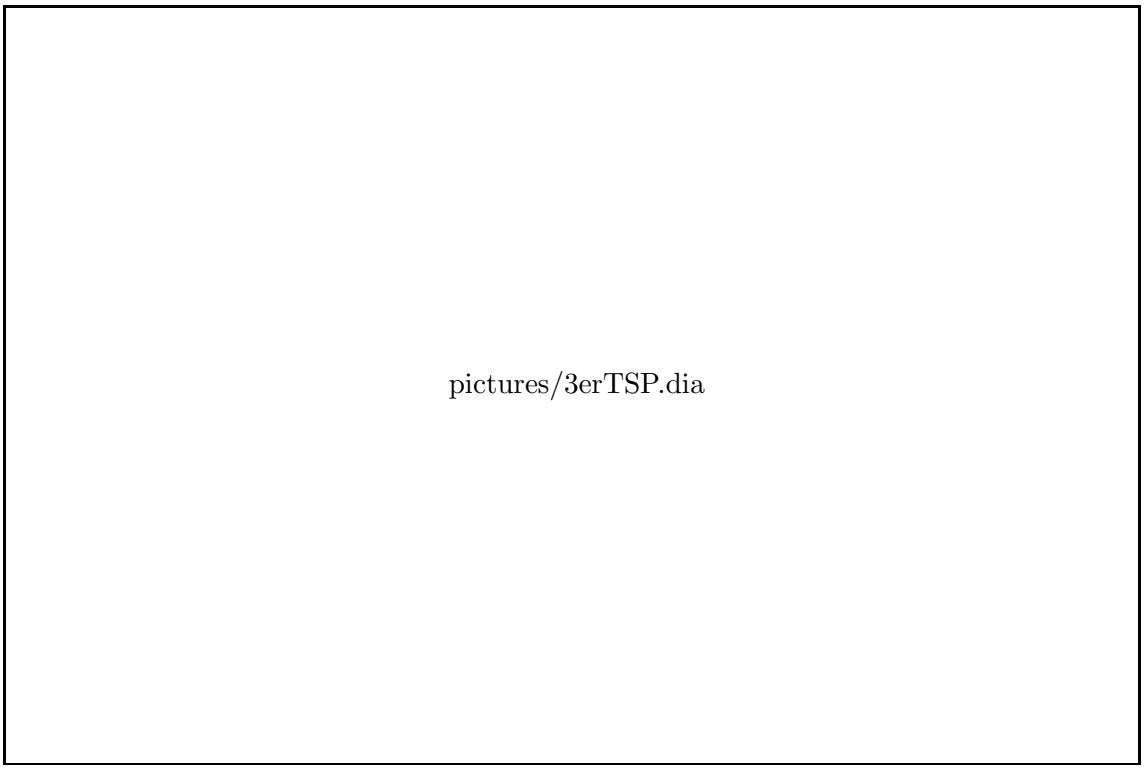

Abbildung D.13: Lösung eines Travelling Salesman Problems, dargestellt mit dem *Wavefront Advanced Visualizer*.

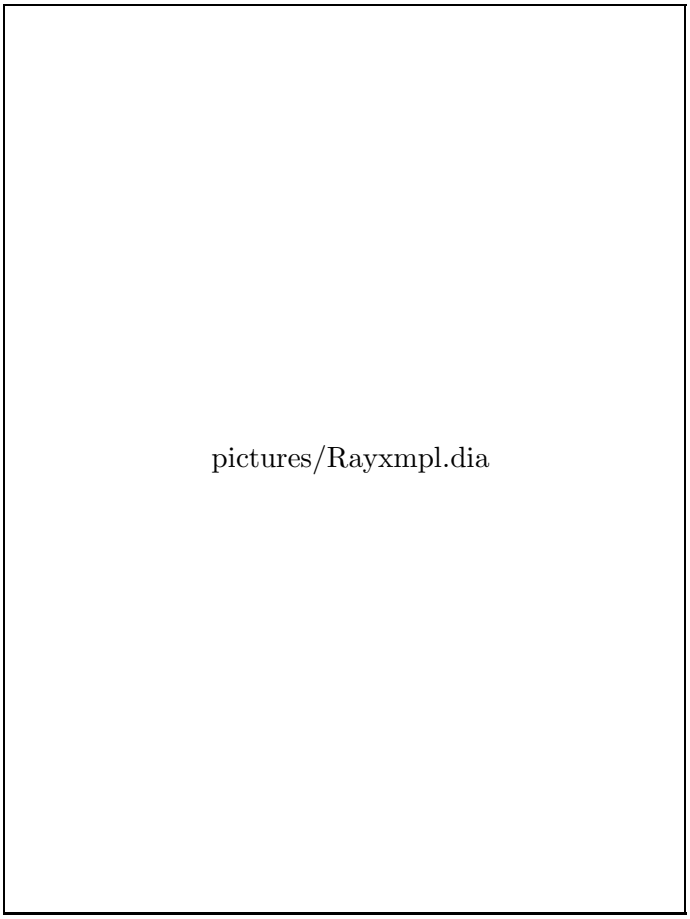

Abbildung D.14: Mit *Rayshade* erzeugtes Bild.

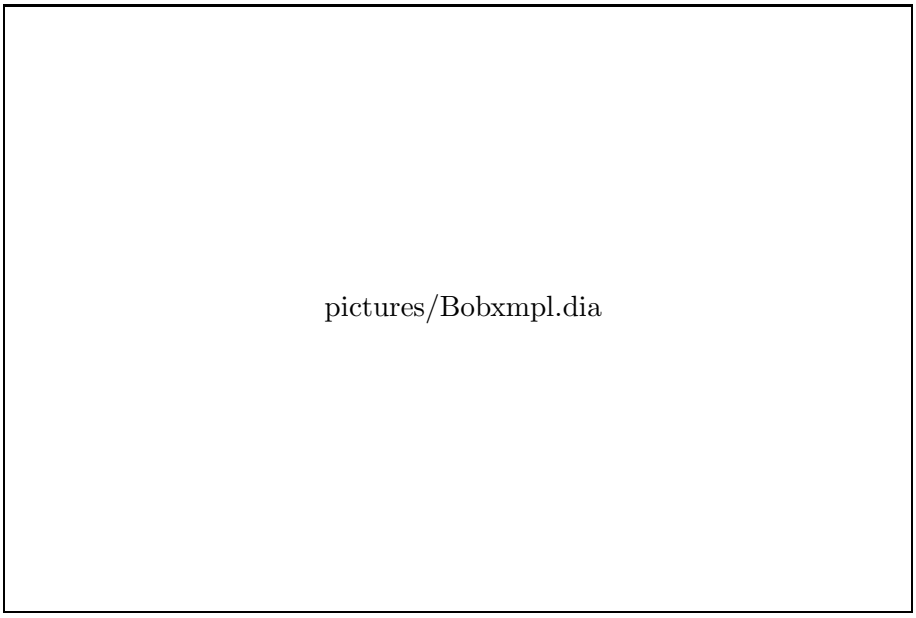

Abbildung D.15: Darstellung eines Schädels auf Basis von CT-Daten und eines berechneten Temperaturfeldes; Bilderzeugung mit dem Volumen-Renderer *BoB*.

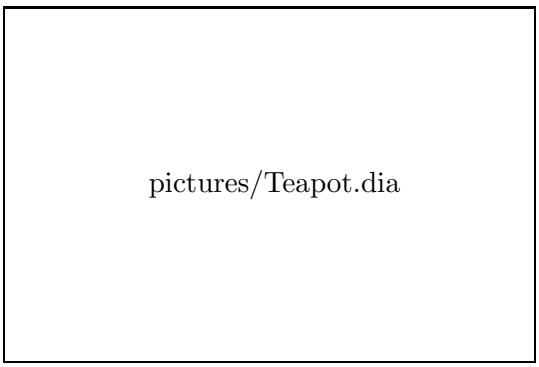

Abbildung D.16: Die berühmte Teekanne (`teapot.ppm`); vergleiche Kapitel 3.2.7.

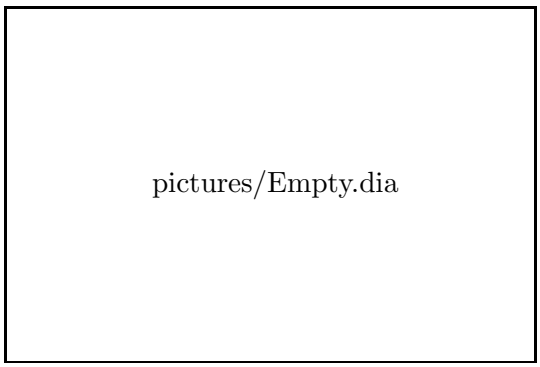

Abbildung D.17: Teekanne entfernt (`empty.rle`).

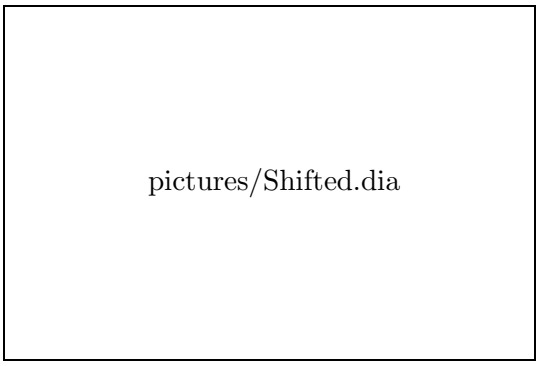

Abbildung D.18: Die Teekanne an neuer Position (`shifted.rle`).

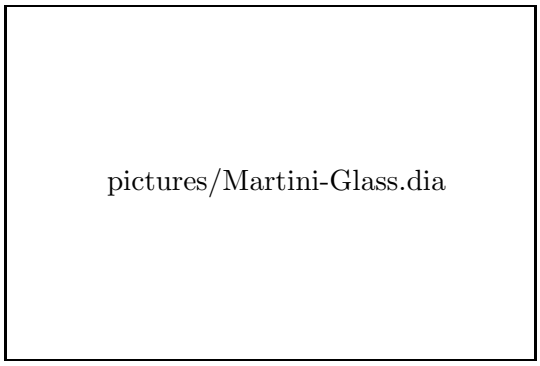

Abbildung D.19: Das Martini-Glas (`martini.rgb`).

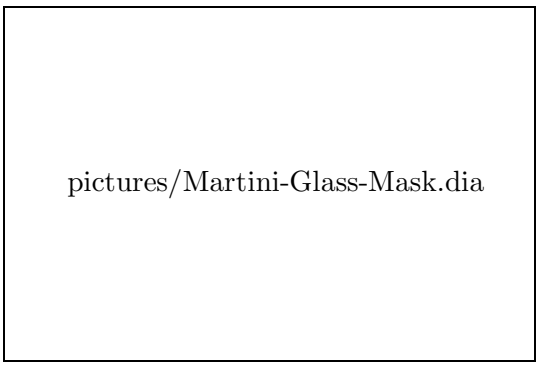

Abbildung D.20: Alle Pixel mit einem Grauwert > 0 sind weiß (`mask.xbm`).

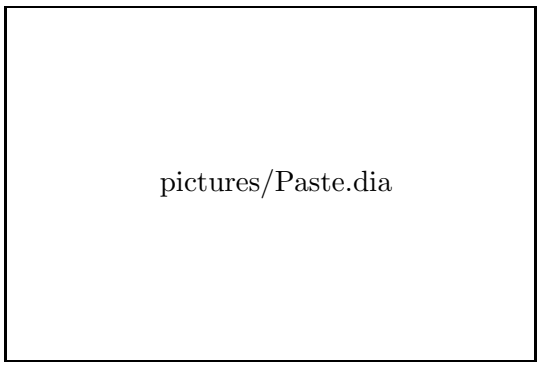

Abbildung D.21: Skaliertes Martini-Glas zur Teekanne kopiert (`paste.rle`).

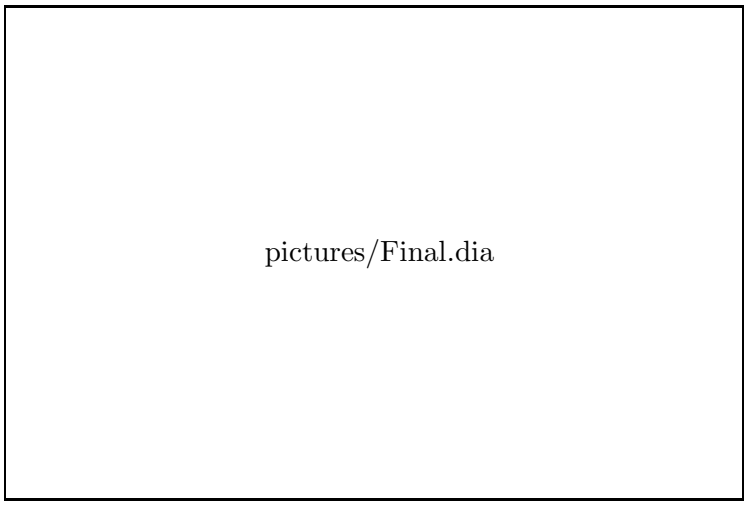

Abbildung D.22: Mit Hilfe der Rastertools komponiertes Bild (`final.rle`).

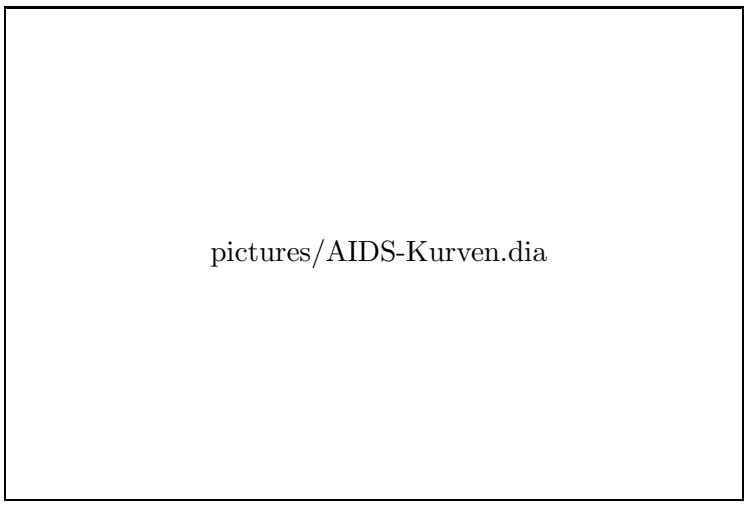

Abbildung D.23: Kurvenplots auf einem Dia.

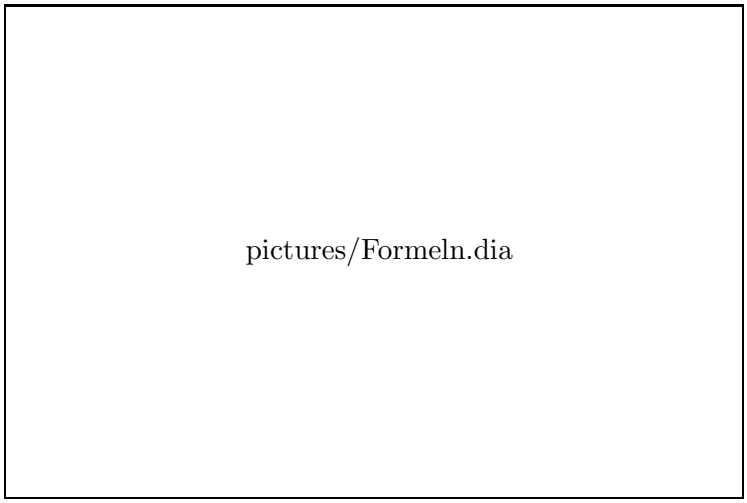

Abbildung D.24: Formeln auf einem Dia.

Literaturverzeichnis

[1] Adobe Systems Incorporated: *PostScript Language Reference Manual*, 2. Auflage, Addison-Wesley Publishing Company, 1990.

[2] Adobe Systems Incorporated: *PostScript Language Reference Manual*, Addison-Wesley Publishing Company, 1986.

[3] Adobe Systems Incorporated: *PostScript Language Tutorial and Cookbook*, Addison-Wesley Publishing Company, 1986.

[4] J. Albers: *Interaction of Color*, DuMont Schauberg, Köln, 1970.

[5] J. Arvo (ed.): *Graphics Gems II*, Academic Press, Boston, 1991.

[6] J. Bechlars, R. Buhtz: *GKS in der Praxis*, Springer Verlag, Berlin, 1986.

[7] S. Bohus: *CLRpaint 1.01*, Release April 1993, Centre for Landscape Research, University of Toronto, 1993

[8] K.W. Brodlie, L.A. Carpenter, R.A. Earnshaw, J.R. Gallop: R. J. Hubbold, A. M. Mumford, C. D. Osland, P. Quarendon, (eds),*Scientific Visualization*, Springer Verlag, Berlin, 1992.

[9] K. Chin-Purcell,*Bob, Manual*, Minnesota Supercomputer Center Inc., 1992, Mail to gvlware@ahpcrc.umn.edu.

[10] H.J. Durrett (ed.): *Color and the Computer*, Academic Press, Boston, 1987.

[11] D.S. Dyer: *A Dataflow Toolkit for Visualization*, IEEE Computer Graphics and Applications, 10 (4), 1990, pp.60-69.

[12] R.A. Earnshaw, N. Wiseman: *An Introductory Guide to Scientific Visualization*, Springer Verlag, Berlin, 1992.

[13] J. Encarnação, W. Straßer: *Computer Graphics, Gerätetechnik, Programmierung und Anwendung graphischer Systeme*, 2. Auflage, Oldenbourg Verlag, 1986.

[14] W. Eifler: *Video Aufzeichnungs Service des ZIB*, ZIB, 1993.

[15] *Figaro+ 3.0 Desk Reference – C Binding*, Liant Software Corp., Sept. 1991, San Diego.

[16] *Figaro+ 3.0 Peripheral Support Option 3.0 – User's Guide*, Liant Software Corp., Aug. 1991, San Diego.

[17] *Figaro+ 3.0 Reference Manual – C Binding*, Vols. 1 and 2, Liant Software Corp., June 1991, San Diego.

[18] *Figaro+ 3.0 Reference Manual – Fortran Binding*, Vols. 1 and 2, Liant Software Corp., June 1991, San Diego.

[19] J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes: *Computer Graphics* Principles and Practice, 2. Auflage, Addison-Wesley Publishing Company, 1990.

[20] T. Gaskins: *PHIGS Programming Manual*, O'Reilly & Associates, 1992.

[21] C. Gerthsen, H.O. Kneser, H. Vogel: *Physik: Ein Lehrbuch zum Gebrauch neben Vorlesungen*, Springer, Berlin, 1977.

[22] *Getting Started with SunPHIGS*, Sun Microsystems, Inc., Okt. 1991, Mountain View.

[23] A.S. Glassner (ed.): *Graphics Gems*, Academic Press, Boston, 1990.

[24] A.S. Glassner: *Ray Tracing*, Academic Press, London, 1989.

[25] O.-J. Grüsser, U. Grüsser-Cornehls: *Physiologie des Sehens*, in R. F. Schmidt (ed.), Grundriß der Sinnesphysiologie, Springer, Berlin, 1976.

[26] J. Gulbins: *Desktop Publishing mit Framemaker*, Springer, 1992.

[27] R. Hall: *Illumination and Color in Computer Generated Imagery*, Springer Verlag, New York, 1989.

[28] D. Heller: *Motif Programming Manual*, The X Window System Series, Volume 6, Motif Edition, O'Reilly & Associates, 1991.

[29] L.R. Henderson, A.M. Mumford: *The Computer Graphics Metafile*, Computer Graphics Standard Series, Butterworth, 1990.

[30] T.L.J. Howard, W.T. Hewitt, R.J. Hubbold, K.M. Wyrwas: *A Practical Introduction to PHIGS and PHIGS PLUS*, Addison Wesley, 1991.

[31] *IRIS Explorer Module Writer's Guide*, Silicon Graphics, Inc., 1993.

[32] *IRIS Explorer User's Guide*, Silicon Graphics, Inc., 1993.

[33] C.E. Kolb: *Rayshade User's Guide and Reference Manual*, Draft 0.4, Princeton University, 1992.

[34] H. Kopka: *LaTeX, Eine Einführung*, 3. Auflage, Addison-Wesley GmbH, 1991.

[35] L. Kosko (ed.): *PHIGS Reference Manual*, O'Reilly & Associates, 1992.

[36] D. Kotz: *LaTeXand GNUPLOT Plotting Program*, contained in the official GNUPLOT distribution [72], 1991.

[37] T.L. Kunii (ed.): *Visual Computing*, Springer Verlag, Tokyo, 1993.

[38] J. Langendorf, O. Paetsch: *GRAZIL – Beschreibung der Version 6.0 des Plotpaketes*, ZIB-TR 1/93.

[39] B.H. McCormick, T.A. DeFanti, M.D. Brown (eds.): *Visualization in Scientific Computing*, Computer Graphics 21(7), November 1987.

[40] *NAG Graphics Library Handbook - Mark 3*, The Numeriacal Algorithms Group Limited, 1990.

[41] J. Neider, T. Davis, M. Woo : *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, Release 1, Addison Wesley, 1993.

[42] A. Nye, T. O'Reilly: *X Toolkit Intrinsics Programming Manual, Motif Edition*, The X Window System Series, Volume 4, Motif Edition, O'Reilly & Associates, 1990.

[43] A. Nye, T. O'Reilly: *X Toolkit Intrinsics Programming Manual*, The X Window System Series, Volume 4, O'Reilly & Associates, 1990.

[44] A. Nye: *Xlib Programming Manual, und Xlib Reference Manual*, The X Window System Series, Volume 1 and 2, O'Reilly & Associates, 1988.

[45] T. O'Reilly, V. Quercia: *X Window System User's Guide, Motif Edition*, The X Window System Series, Volume 3, Motif Edition, O'Reilly & Associates, 1990.

[46] T. O'Reilly, V.Quercia, L.Lamb: *X Window System User's Guide*, The X Window System Series, Volume 3, O'Reilly & Associates, 1990.

[47] OpenGL Architecture Review Board,*OpenGL Reference Manual: The Official Reference Document for OpenGL*, Release 1, Addison Wesley, 1993.

[48] *PEXlib Specification and C Language Binding*, Version 5.1P – Public Review Draft, in The X Resource, Special Issue B, Sept. 1992, O'Reilly & Associates, 1992.

[49] M.S. Peercy: *Linear Color Representations for Full Spectral Rendering*, SIGGRAPH Proceedings 1993, pp.191-198.

[50] E. Pepke, J. Lyons: *SciAn User's Manual*, May 1993, Supercomputer Computation Research Institute, Florida State University.

[51] C. Prichard, L. Hollenhorst-Rose, D. Ramey: *The Advanced Visualizer 3.0, User's Guide Volume 1-3*, 2nd Edition, Wavefront Technologies Inc., 1992.

[52] J. Rasure, D. Agiro, T. Sauer, C. Williams:

[53] D.F. Rogers: *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, 1985.

[54] L.H. Rose: *The Composer 2.0 Release Notes*, Wavefront Technologies Inc., 1992.

[55] L.H. Rose: *The Composer 2.0 User's Guide*, 3nd Edition, Wavefront Technologies Inc., 1992.

[56] R.W. Scheifler: *X Protocol Reference Manual*, The X Window System Series, Volume 0, O'Reilly & Associates, 1989.

[57] P.S. Strauss, R. Carey: *An Object-Oriented 3D-Graphics Toolkit*, Computer Graphics 26 (2), 1992, pp.341-349.

[58] *SunPHIGS 2.0 Extensions Reference Manual*, Sun Microsystems, Inc., Okt. 1991, Mountain View.

[59] *SunPHIGS 2.0 Porting Guide*, Sun Microsystems, Inc., Okt. 1991, Mountain View.

[60] *SunPHIGS 2.0 Programming Guide*, Sun Microsystems, Inc., Okt. 1991, Mountain View.

[61] *SunPHIGS 2.0 Reference Manual*, Vols. 1 and 2, Sun Microsystems, Inc., Okt. 1991, Mountain View.

[62] S. Satanthavibul: *xfig 2.1, Manpages*, University of Texas at Austin, USA, 1993.

[63] E.R. Tufte: *Envisioning Information*, Graphics Press, Cheshire (Co), USA, 1990.

[64] *UniChem 2.0 User' Guide APG-5500 2.0*, Cray Research Inc., 1993

[65] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel,J. Vroom, R. Gruwitz, A. van Dam: *The Application Visualization System: A Computational Environment for Scientific Visualization*, IEEE Computer Graphics and Applications, 9 (4), 1990, pp.30-42.

[66] *Using PHIGS with Figaro+*, Template Graphics Software, Inc., 1990, San Diego.

[67] J. Watkinson: *The Art of Digital Video*, Focal Press, London 1990.

[68] A. Watt: *Fundamentals of Three-Dimensional Computer Graphics*, Addison-Wesley Publishing Company, 1990.

[69] J. Webers: *Handbuch der Film- und Videotechnik*, Franzis-Verlag, München 1988.

[70] M.J. Wichura: *The PiCTEX Manual*, Publications for the TEX Community, Number 6, M. Pfeffer & Co., NY 1987.

[71] A. Wierse, M. Rumpf: *GRAPE – Eine objektorientierte Visualisierungs- und Numerikplattform*, Informatik Forsch. Entw. 7, 1992, 145-151.

[72] T. Williams, C. Kelley: *GNUPLOT 3.2, Manual, An Interactive Plotting Program*, 1993, Mail to info-gnuplot@ames.arc.nasa.gov.

[73] R. Wunderling, *AGIL*, Internal Report, ZIB, 1992.

[74] G. Wyszecki, W. Stiles: *Color Science: Concepts and Methods, Quantitative Data and Formulae*, second edition, Wiley, New York, 1982.

[75] *X Toolkit Intrinsics Reference Manual*, The X Window System Series, Volume 5, Motif Edition, O'Reilly & Associates, 1990.

Index