Modulare Software für die höherdimensionale Bildverarbeitung

Helmut Herrmann[1] *und Bernd Jähne*[2]

[1]AEON Verlag & Studio Walter H. Dorn
Digital Image Processing
Fraunhoferstr. 51B, D-63454 Hanau
Tel.: 06181/23525, Fax: 06181/257954
E-mail: helmut.herrmann@aeon.de
[2]Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg
Im Neuenheimer Feld 368, D-69120 Heidelberg
Tel.: 06221/54 8827, Fax: 06221/54 8850
E-mail: bernd.jaehne@iwr.uni-heidelberg.de

Zusammenfassung

Software für die Entwicklung von Bildverarbeitungsanwendungen muß weitgehend plattformübergreifend, modular und offen aufgebaut sein, wenn sie nachhaltig zu einem erfolgreichen Einsatz in komplexen Aufgabenstellungen wie der 3D-Bildverarbeitung führen soll. Dies folgt aus der Tatsache, daß nicht ein einzelner Anbieter alles Erforderliche liefern kann, sei es Hardware oder Software. Es müssen also mehrere Komponenten unterschiedlicher Hersteller zusammenspielen, und es sollte bei Bedarf ein Wechsel der Plattform möglich sein. Anhand des erweiterten Konzeptes der Bildverarbeitungssoftware heurisko werden diese Prinzipien erläutert und Beispiele zu ihrer Nutzung gezeigt.

1 Einleitung

In diesem Beitrag werden die Prinzipien portabler und modularer Software für die Bildverarbeitung anhand des Softwarepaketes heurisko beleuchtet. heurisko, das nicht speziell für die 3D-Bildverarbeitung entwickelt wurde, stellt ein universelles Entwicklungsinstrument dar und kann neben 3D-Bilddaten auch Bildsequenzen und Mehrkanalbilder verarbeiten. Die Software wurde und wird in einer Kooperation der Firma AEON mit Prof. B. Jähne von der Universität Heidelberg entwickelt und wird sowohl von Universitätsinstituten als auch in der Industrie eingesetzt. Das Paket stellten wir bereits auf dem 4. ABW-Workshop vor, dort mit dem Schwerpunkt auf der Algorithmik [*Jähne, Herrmann*, 1997]. Das hier gewählte Thema geht einher mit der gegenwärtigen Weiterentwicklung, die im zweiten Quartal dieses Jahres abgeschlossen sein wird. Leser, die mit heurisko bereits arbeiten, sollten deshalb beachten, daß ihre aktuelle Version noch nicht dem hier Dargestellten entspricht.

2 Anforderungen an eine Bildverarbeitungssoftware

Aus der Sicht des Anwenders eines Softwarepaketes zur Entwicklung von Bildverarbeitungsanwendungen sind folgende Eigenschaften eines solchen Paketes wünschenswert:

Problemorientierte Programmiersprache: In normalen Programmiersprachen gibt es weder Bildobjekte noch Funktionen, die Bilder verarbeiten. Deshalb kann man mit Ihnen in der Bildverarbeitung auch nicht problemorientiert programmieren, sondern muß sich auf eine tiefere Stufe begeben. Das erhöht den Programmieraufwand und die Fehleranfälligkeit des Programmes. Eine Hochsprache für Bildverarbeitung sollte deren relevante Datentypen und Datenstrukturen kennen

doi: 10.5281/zenodo.14326

und daran angepaßte Operatoren anbieten. Die Faltung eines Bildobjektes z. B. sollte unabhängig von den Objektdimensionen ein einziger Befehl sein, wobei die Faltungsmaske ein Parameter der entsprechenden Funktion ist.

Offenes und modulares System mit Schnittstellen für Erweiterungen: Da man mit einem Programmpaket, wie es gerade skizziert wurde, sicher nicht allen Bedürfnissen gerecht wird und nicht jede benötigte oder gewünschte Funktionalität bietet, darf das Entwicklungssystem nicht abgeschlossen sein, sondern muß Erweiterungsmöglichkeiten aufweisen. Beispiele für Erweiterungen sind effiziente neue Algorithmen und die Anbindung neuer Hardware wie Framegrabber oder spezielle Prozessoren.

Portable Anwendungen: Jeder weiß, wie schnell die Hardwareentwicklung voranschreitet. Das bringt es mit sich, daß hardwarenahe Programme ständig an die neuen Gegebenheiten angepaßt werden müssen. Wenn keine besonderen Gründe wie z. B. höchste Performance für eine Spezial-Hardware sprechen, strebt man deshalb eine hardware- und auch betriebssystemunabhängige Programmierung an. Das sollte aber möglichst wenig zu Lasten der Leistungsfähigkeit gehen.

Mehrdimensionale Datenobjekte: Bei vielen Bildverarbeitungspaketen, welche nicht nur für spezielle Anwendungen geschaffen wurden, ist nach wie vor die 2D-Bildverarbeitung der Standard. Es gibt jedoch immer mehr Anwendungen, auch bei der Rekonstruktion von 3D-Objekten, die sich Bildsequenzen zunutze machen. Es werden hierbei nicht nur zwei aufeinanderfolgende Bilder der Sequenz betrachtet oder Korrelationen vorgenommen, sondern die Zeit kommt als echte weitere Koordinate hinzu. Darüber hinaus kann es wünschenswert sein, Objekte in verschiedenen Kanälen zu betrachten, z. B. mit in verschiedenen Spektralbereichen unterschiedlich empfindlichen Sensoren. So erhält man quasi eine zusätzliche Dimension. Je nachdem, ob man sich mit 2D-Bildern oder Volumendaten beschäftigt, ergeben sich damit vier- oder fünfdimensionale Objekte. Obwohl die Rechengeschwindigkeit von Standard-Hardware heute dazu noch nicht ausreicht, sollte eine Software entsprechende Objekte verarbeiten können.

Problemangepaßte Datentypen: Hier muß der Speicherplatzbedarf gegen die Rechengenauigkeit und den benötigten Zahlenbereich abgewogen werden. Auch die Rechengeschwindigkeit spielt eine Rolle bei der Wahl des Datentyps. Zur Digitalisierung von langen Bildsequenzen wird man beispielsweise 8-Bit-Pixel verwenden. Bei anschließenden Auswertungen, die nach Vorverarbeitungen und eventuell nur auf interessierenden Teilbereichen stattfinden, wird man dann etwa wegen der gewünschten Genauigkeit auf Gleitkommazahlen mit 32 Bit oder mehr übergehen. Gibt es einen eigenen Datentyp für Binärbilder, wird man diese besonders effektiv speichern und schnell verarbeiten können. Für Transformationen wie die Fouriertransformation sind auch noch komplexe Datentypen hilfreich.

Effiziente Algorithmen: Nicht zuletzt spielt die Performance eine große Rolle, denn sie bestimmt wesentlich, welche Probleme überhaupt in der Praxis mit Bildverarbeitung gelöst werden können. Effiziente Algorithmen sind auch der Schlüssel auf dem Weg von teuren Spezialrechnern zu günstiger Standard-Hardware. Letztere erleichtert eine portable Softwareentwicklung. In heurisko wurde deshalb auf eine möglichst effiziente Implementierung aller Algorithmen geachtet. Die Grundlagen dazu sind in *Jähne* [1997a] und *Jähne* [1997c] beschrieben.

Die nächsten Kapitel beschäftigen sich insbesondere mit der zweiten Anforderung dieser Aufstellung, nämlich den Möglichkeiten, die sich mit einer modularen und erweiterbaren Software bieten.

3 Modularität und Erweiterbarkeit

Das Blockdiagramm in Abb. 1 stellt die modulare Architektur heuriskos dar. heurisko besteht im wesentlichen aus drei Teilen, der Benutzerschnittstelle, einem Interpreter/Compiler und dem Kern mit den eigentlichen Bildverarbeitungsfunktionen, die über Schnittstellen miteinander kommunizieren.

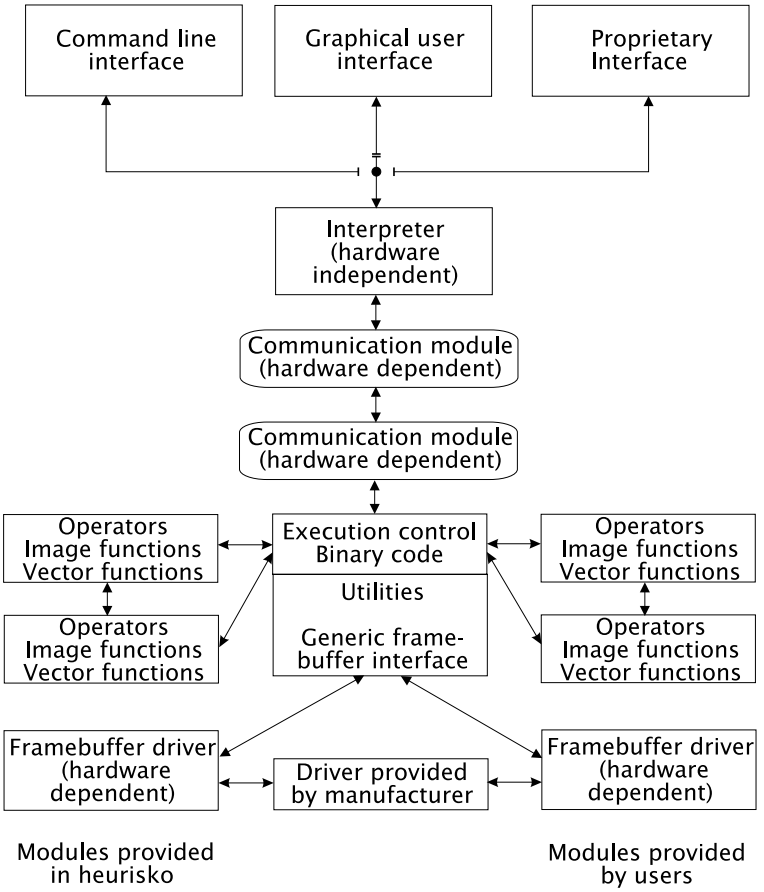

Abbildung 1: Blockdiagramm des modularen und hierarchischen Aufbaus des Softwarepakets heurisko

Der Interpreter kann mehrere Kerne steuern. Somit ist in dem Ansatz paralleles und verteiltes Rechnen zwar noch nicht realisiert, aber bereits enthalten. Der Kern ist selbst wieder in Funktionsgruppen unterteilt, einmal hierarchisch nach Operatoren, Bild- und Vektorfunktionen, aber auch nach der Art der Operationen. Bislang besteht heurisko nur aus zwei Moduln, nämlich der Benutzeroberfläche und dem Paket aus Interpreter/Compiler und Kern. In Kürze wird sich die Modularisierung auch äußerlich in getrennten Moduln — unter Windows als DLLs und unter UNIX als Shared Libraries — niederschlagen. Dann wird man nur diejenigen Teile heuriskos installieren und laden müssen, die man für eine bestimmte Anwendung auch benötigt.

Welche Änderungen und Erweiterungen kann nun jemand, der mit heurisko Bildverarbeitungsanwendungen entwickelt, vornehmen? Es bestehen dazu im wesentlichen drei Möglichkeiten, die sich beliebig kombinieren lassen.

Die erste Möglichkeit besteht darin, bereits bekannte Operatoren zu neuen Operatoren zu gruppieren, wobei auch Kontrollstrukturen wie if/else und verschiedene Schleifen zu Verfügung stehen. Dies entspricht sowohl dem Funktionskonzept als auch dem Unterprogramm- oder Modulkonzept anderer Programmiersprachen. An dieser Stelle soll zum vertieften Verständnis der typische Ablauf einer heurisko-Anwendung skizziert werden. Eine heurisko-Anwendung wird in einer ASCII-Textdatei, welche Workspace genannt wird, gespeichert. Der gewünschte Workspace wird zu Beginn einer Anwendung eingeladen und dem Interpeter/Compiler übergeben. Dieser Modul übersetzt den Workspace in einen schnellen Binärcode und gibt jenen an den Kern weiter. (Diese Technik haben übrigens jüngst auch die Entwickler des Tcl/Tk-Paketes in der neuen Version 8.0 angewandt. Interessierte finden dazu mehr unter http://www.sunlabs.com/research/tcl/8.0.html.) Hat man das Konzept, Operatoren zu

komplexeren Einheiten in Form eines neuen Operators zusammenzufassen, intensiv genutzt, benötigt der Interpreter von da ab nur einen einzigen Befehl, um eine komplexe Verarbeitung in Gang zu setzen. Die Zeit zur Interpretation des einen Befehls läßt sich dann vernachlässigen. heurisko arbeitet demnach in zwei Phasen; in der Initialisierungsphase wird der Workspace interpretiert und übersetzt, und in der Hauptphase wird der schnelle Binärcode ausgeführt. Nutzt man keine der beiden folgenden Möglichkeiten, sondern nur die hier beschriebene, kann man mit heurisko-Workspaces Bildverarbeitungsanwendungen ohne eine einzige Zeile C-Code oder einer anderen Programmiersprache entwickeln.

Die zweite Möglichkeit, heurisko an die eigenen Bedürfnisse anzupassen, betrifft die Benutzeroberfläche. Die offengelegte Schnittstelle zum Interpreter/Compiler erlaubt das Abkoppeln des heurisko-Oberflächenmoduls. Stattdessen kann entweder eine eigene Benutzungsschnittstelle eingefügt oder aber heurisko in ein anderes Programmpaket integriert werden. Letzteres ist z. B. dann der Fall, wenn die Bildverarbeitung nur einen Teil einer Anlagensteuerung ausmacht. Die Schnittstelle ist sehr einfach zu benutzen, da im wesentlichen nur eine einzige Funktion benötigt wird, der beliebig lange Kommandos in der heurisko-Skriptsprache als Parameter übergeben werden.

Die dritte und mächtigste Art, heurisko zu erweitern, ist die Bereitstellung eigener heurisko-Zusatzmoduln, welche, wie bereits oben erwähnt, in DLLs oder Shared Libraries verpackt und bei Bedarf von heurisko geladen werden. Die Schnittstelle zwischen heurisko und Fremdmoduln ist so konzipiert, daß heurisko vorher überhaupt nicht wissen muß, welche Moduln der Anwender irgendwann einmal laden will. Man muß heurisko nur zur gegebenen Zeit mitteilen, daß man einen bestimmten Modul benutzen möchte. Nach dem Laden erfragt heurisko über eine Funktion, welche jeder fremde Modul exportieren muß, nach einer Referenz auf eine standardisierte Struktur, in der die anderen von dem Modul angebotenen Funktionen beschrieben sind. heurisko wertet diese Struktur aus und bietet nun entsprechende Operatoren zur Verwendung in der Skriptsprache an. Es gab zwar schon in der ersten Windows-Version heuriskos einige Beispiele für dynamisch eingebundene Fremdmoduln, aber diese mußten bereits vom heurisko-Hersteller im heurisko-Code explizit berücksichtigt werden. Es handelte sich dabei um die Framegrabberuntertützung, bei der dem Anwender nicht die zu verwendende Hardware vorgeschrieben werden konnte. Das neue Konzept geht einen entscheidenden Schritt weiter und weist folgende wichtige Vorteile auf:

- Die selbstentwickelten Moduln müssen nicht mit heurisko gelinkt werden, und heurisko muß die später einmal vom Anwender bereitgestellten Moduln überhaupt nicht kennen.
- Nach einem Update irgendeines Moduls läßt sich nach dessen Installation sofort weiterarbeiten.
- Zusatzmoduln können problemlos an andere heurisko-Anwender weitergegeben werden, vorausgesetzt, deren heurisko-Version ist kompatibel mit der des Zusatzmodulentwicklers. Damit können heurisko-Anwender ihre Entwicklungen auch anderen zugänglich machen.
- Zusatzmoduln lassen sich zu einem beliebigen Zeitpunkt laden und auch wieder freigeben.

4 Beispiele

Im folgenden soll an drei Beispielen demonstriert werden, wozu das erweiterte Modulkonzept genutzt werden kann.

4.1 Das Softwarepaket ABW-3D

Das Softwarepaket ABW-3D, welches ABW zur Unterstützung der Arbeit mit seinen Streifenprojektoren anbietet, ist für die Teilnehmer des Workshops naturgemäß ein interessantes Beispiel. Diese Software wurde in C++ geschrieben und hatte bei seiner Entwicklung nichts von heurisko gewußt. heurisko selbst ist aus zwei Gründen bislang in C geschrieben. Zum einen gab es zum Zeitpunkt der Entscheidung keinen verabschiedeten C++-Standard, und zum anderen ist C auch heute noch wesentlich wei-

ter verbreitet als die objektorientierte Programmiersprache. heurisko ist bis auf Ausnahmen, welche die Unterstützung bestimmter Hardware und die graphische Benutzerschnittstelle betreffen, strikt in ANSI-C codiert, weil eine der Anforderungen die Portabilität ist. (heurisko wurde bereits auf mehrere UNIX-Plattformen portiert, wobei jeweils nur wenige Stunden benötigt wurden. Dazu sei angemerkt, daß die portable Benutzerschnittstelle unter X-Windows mit dem Tcl/Tk-Paket erstellt wurde.)

ABW und AEON realisierten die Kooperation ihrer Software, als sie den Auftrag eines gemeinsamen Kunden hatten. Schon nach dem ersten Informationsaustausch wurde klar, daß beide Pakete auf Funktionen des jeweils anderen Paketes Zugriff haben sollten. ABW-3D sollte sich die umfangreiche Framegrabberunterstützung heuriskos zunutze machen können, und heurisko sollte, das war das Hauptziel, das Arbeiten mit ABW-3D innerhalb der Workspaces ermöglichen. Auf einem Treffen der zuständigen Entwickler der beiden Firmen wurde kurzerhand eine Schnittstelle definiert und diese innerhalb weniger Tage programmiert, getestet und ausgeliefert. ABW-3D erhielt dabei ein C-Interface und heurisko einen Modul, der die von ABW-3D angebotenen Funktionen für heurisko aufbereitet. Im folgenden sieht man einen Ausschnitt eines heurisko-Workspaces mit Aufrufen von ABW-3D-Funktionen, deren Namen mit Abw3d anfangen.

heurisko-Workspace mit ABW-3D

```
# Framegrabber-Objekt
framebuf fg {
    string type,
    string config,
    float range[2]
};
fg.type = "pc_eye2"; # ELTEC PC_EYE2
fg.config = "eye8g.dat";
fg.range = {0,255};

# Bilder
byte mask[576][748]; # Maske
byte grey[576][748]; # Grauwertbild
float xyz[3][576][748]; # Koordinaten

# Konstanten
XYZMG:=46; # x,y,z, Maske und Grauwerte

# Initialisierung
Abw3dInit(fg,1); # Grabber fg benutzen
# Parameter und Kalibrierdaten
Abw3dReadParam("abw3d.inp");
Abw3dReadCal("abw3d.cal");
```

```
# Operatoren
operator objfeatures();
...
endoperator;

operator showresults();
...
endoperator;

operator measure();
    Abw3dMeasure();
    xyz[0]=Abw3dGetX();
    xyz[1]=Abw3dGetY();
    xyz[2]=Abw3dGetZ();
    mask=Abw3dGetMask();
    grey=Abw3dGetGrey();
    objfeatures();
....showresults();
endoperator;
...
# Aufruf des Operators measure zum
# Messen, Auswerten und Darstellen
measure();
...
```

4.2 Der erweiterte Prozessorinstruktionssatz MMX

Obwohl für heurisko die Hardware-Unabhängigkeit eine der Hauptanforderungen ist, wird die Multimedia-Instruktionssatzerweiterung der Intel-Prozessoren unterstützt. Es wurden jedoch nur solche Funktionen in MMX-Assembler geschrieben, die bereits in C implementiert waren. Von daher laufen alle heurisko-Anwendungen unabhängig davon, ob tatsächlich ein MMX-Prozessor vorhanden ist oder nicht. Für die Implementierung in MMX sprachen mehrere Gründe.

- Bildverarbeitung kann nie schnell genug sein; deshalb sollten alle Leistungsreserven genutzt werden.

- MMX darf man wegen der weiten Verbreitung und der durch Intel gegebenen längerfristigen Perspektiven durchaus als Standard-Hardware ansehen. MMX wird außerdem auch von Intel-Konkurrenten unterstützt.
- Um heurisko zu beschleunigen, muß nur ein sehr geringer Anteil des Codes an MMX angepaßt werden. Dies liegt an der strengen hierachischen Gliederung, so daß letztlich alle Bildverarbeitungsoperationen eine der Vektorfunktionen aufrufen.
- Für Integer-Operationen, welche besonders bei der Bildvorverarbeitung eine Rolle spielen und meistens noch auf den gesamten Bilddaten arbeiten, ist abhängig von den Cache- und Speicherverhältnissen und vom Datentyp ein Beschleunigungsfaktor zwischen 2 und 10 zu erreichen. Ergebnisse konkreter Messungen mit heurisko findet man unter http://www.aeon.de/heurisko.

MMX-Code und normaler Code werden in separaten Moduln untergebracht, welche nach der Identifizierung des Prozessors entsprechend geladen werden. Über die beiden erwähnten Moduln hinaus kann man auch für jeden Prozessortyp einen spezifischen Modul, der mit der zugehörigen Optimierung übersetzt wurde, zur Verfügung stellen.

Wer sich für weitere Details von MMX insbesondere im Zusammenhang mit Bildverarbeitung interessiert, sei auf *Jähne* [1997b] verwiesen.

4.3 Bildverarbeitung am IWR der Universität Heidelberg

Die Bildverarbeitungsgruppe des Interdisziplinären Institutes für Wissenschaftliches Rechnen (IWR) und eine Arbeitsgruppe des Institutes für Umweltphysik der Universität Heidelberg verwenden heurisko als zentrales Werkzeug bei der Entwicklung ihrer Bildverarbeitungsanwendungen. Momentan kooperieren außerdem weitere Institute innerhalb einer von der DFG geförderten Forschergruppe, die sich das Studium dynamischer Prozesse in der Physik, der Physiologie und in der Botanik mit Bildverarbeitung zur Aufgabe gemacht hat [*Jähne et al.*, 1996], [*Jähne et al.*, 1997]. Auch hier ist heurisko zentrales Werkzeug. Aus den bisherigen Forschungsarbeiten sind bereits rund ein Dutzend Zusatzmoduln hervorgegangen, unter ihnen einer zum Einlesen von Satellitendaten des GOME-Instruments, ein anderer mit nichtlinearen Filtern und auch einer für Particle Tracking Velocimetry. Das neue modulare Konzept heuriskos wird die Zusammenarbeit der beteiligten Institute wesentlich erleichtern.

heurisko wird aufgrund seiner Vorteile an der Universität Heidelberg und an anderen Hochschulen auch in Bildverarbeitungspraktika eingesetzt.

Danksagung

Die Autoren danken der Mitwirkung von Peter Geissler, Hermann Lauer, Carsten Leue und Dominik Schmundt (alle IWR, Universität Heidelberg) an der Modularisierung von heurisko.

Literatur

Jähne B, Haußecker H, Hering F, Balschbach G, Klinke J, Lell M, Schmundt D, Schultz M, Schurr U, Stitt M, and Platt U (1996) The Role of Active Vision in Exploring Growth, Transport, and Exchange Processes, *Aktives Sehen in technischen und biologischen Systemen*, Workshop der GI-Fachgruppe 1.0.4 Bildverstehen, Hamburg, 1996.

Jähne B, Herrmann H (1997) Softwarekonzepte und Algorithmen für die 3D-Bildverarbeitung, 4. ABW-Workshop, TA Esslingen, 22.–23.1.1997.

Jähne B (1997a) *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, Boca Raton, 1997.

Jähne B (1997b) SIMD-Bildverarbeitungsalgorithmen mit dem Multimedia Extension-Instruktionssatz (MMX) von Intel, *Automatisierungstechnik, 45*, 453–460.

Jähne B, Haußecker H, Hering F, Platt U, Schurr U, and Stitt M (1997) *Image Sequence Analysis to Investigate Dynamic Processes*, Progress Report Phase I, DFG Forschergruppe, August 1997 (http://klimt.iwr.uni-heidelberg.de).

Jähne B (1997c) *Digitale Bildverarbeitung*, 4. Auflage, Springer-Verlag, Heidelberg, 1997.