Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-97-01

NetGLTool Benutzeranleitung

Thomas Malik

January 1997

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

NetGLTool Benutzeranleitung

Thomas Malik

DFKI-D-97-01

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-9401).

ISSN 0946-0098

Kurzfassung

In diesem Dokument wird ein Visualisierungs-Werkzeug vorgestellt, das über eine Netzwerkverbindung eine Kommandoschnittstelle bereitstellt. Die Kommandosprache bietet Funktionen zur graphischen Ausgabe auf eine Zeichenfläche sowie einige Kommandos zur GUI-Programmierung, z.B. zur Erzeugung von Menüs und einfachen, vordefinierten Dialog-Boxen. Der Schwerpunkt liegt bei den graphischen Fähigkeiten, so enthält das Programm Möglichkeiten, Vektorgraphik, Rasterbilder und gerichtete Graphen in ein frei wählbares Koordinatensystem auszugeben.

Abstract

This document presents a visualization tool, which supplements a command interface via a network connection. The built-in command language offers facilities for graphical output and simple GUI programming, e.g. for the construction of a menu hierarchy and predefined dialog boxes. The emphasis of the command set is on the graphical output facilities, offering the possibility to output vector graphics, bitmap images and directed graphs into a freely definable coordinate system.

Inhaltsverzeichnis

1 Vorwort

Diese Benutzeranleitung umfaßt die Beschreibungen zu NetGLTool, der programmierbaren, netzwerkfähigen Graphikoberfläche, sowie den Komponenten, aus denen diese Oberfläche besteht. Im ersten Abschnitt dieser Anleitung wird beschrieben, welche Motivation hinter der Entwicklung der hier beschriebenen Werkzeuge stand. In den folgenden Kapiteln werden zunächst grundlegende Konzepte der GL-Kommandosprache und die Benutzung des NetGLTool-Servers beschrieben. Es folgen eine Befehlsreferenz mit einer formalisierten Beschreibung der einzelnen GL-Kommandos. Im letzten Teil dieses Handbuches werden die Libraries von NetGLTool beschrieben. Ein Anhang bietet Programmierbeispiele zu den einzelnen Abschnitten.

2 Einleitung, Motivation

Im Umfeld der Projektgruppe OMEGA[1] am DFKI Kaiserslautern entstanden eine Reihe von Anforderungen, die die Entwicklung eines netzwerkfähigen Visualisierungswerkzeuges notwendig gemacht haben.

- Interfacebeschreibung auf hohem Abstraktionsniveau. Die Analyseverfahren werden im allgemeinen von verschiedenen Entwicklern programmiert. Diese Entwickler haben bestimmte Anforderungen an die Visualisierung der von ihren Verfahren erzeugten Daten. Die Visualisierung dieser Daten wurde jedoch bisher von anderen Entwicklern programmiert. Es hat sich gezeigt, daß ein hoher Zeitaufwand für die Kommunikation zwischen diesen verschiedenen Entwicklern notwendig war, nur um die Visualisierung in der gewünschten Form zu realisieren. Zukünftig sollen die Entwickler der Analyseverfahren selber in der Lage sein, auf einfachste Weise die Visualisierung ihrer Daten zu implementieren.
- Klare Trennung von Visualisierung und Analyseverfahren. Die eigentlichen Algorithmen zur Dokumentanalyse sollen auch ohne graphische Ausgabe lauffähig sein.
- Hoher Grad an Portierbarkeit. Durch die Verwendung einer graphischen Beschreibungssprache, die weitgehend vom Fenstersystem unabhängig ist, wird eine Portierung auf andere Standards (z.B. Microsoft Windows oder OS/2) erheblich vereinfacht. Der Aufwand hierbei besteht im wesentlichen darin, NetGLTool auf die Zielmaschine zu portieren. Durch die Netzwerkfähigkeit ist sogar eine Verwendbarkeit in heterogenen Netzen denkbar.
- Lastverteilung auf mehrere Rechner/CPU's. Da die Visualisierung selbst eine beträchtliche Menge an Resourcen benötigen kann (sehr große Rasterbilder, große Mengen an graphischen Primitiven) ist eine Aufteilung zwischen Verfahren und Visualisierung auf mehrere Rechner sinnvoll und gefordert. Dies macht die Netzwerkfähigkeit erforderlich.
- Mehrere gleichzeitig laufende Algorithmen sollen die Oberfläche für gemeinsame Ausgaben nutzen können. Die Unterteilung der Dokumentenanalyse in mehrere separate, konkurrierende Prozesse macht dies notwendig.

Diese Anforderungen haben eine Eigenentwicklung notwendig gemacht; um den Entwicklungsaufwand im Rahmen zu halten, basiert diese auf bestehenden Standards (Motif, X11, BSD sockets).

3 Allgemeine Konzepte

NetGLTool ist ein Netzwerkserver, der von einem oder mehreren Client-Programmen Befehle in einer Graphik-orientierten Kommandosprache empfängt und abarbeitet. Wenn im folgenden vom "Server" die Rede ist, so ist hiermit das Programm `ngltool` gemeint.
Bei der Benutzung von Netzwerkdiensten im Internet wird jeder dieser Dienste durch Angabe eines Rechner-Knotens (der IP-Addresse) und einer Port-Nummer spezifiziert. Eine Port-Nummer legt also einen bestimmtem Netzwerkdienst auf einem vorgegebenem Rechner fest; manche Internet-Dienste (z.B. File Transfer Protocol ftp, Remote Login) benutzen einen

[1] Office Mail Expert for Goal directed Analysis

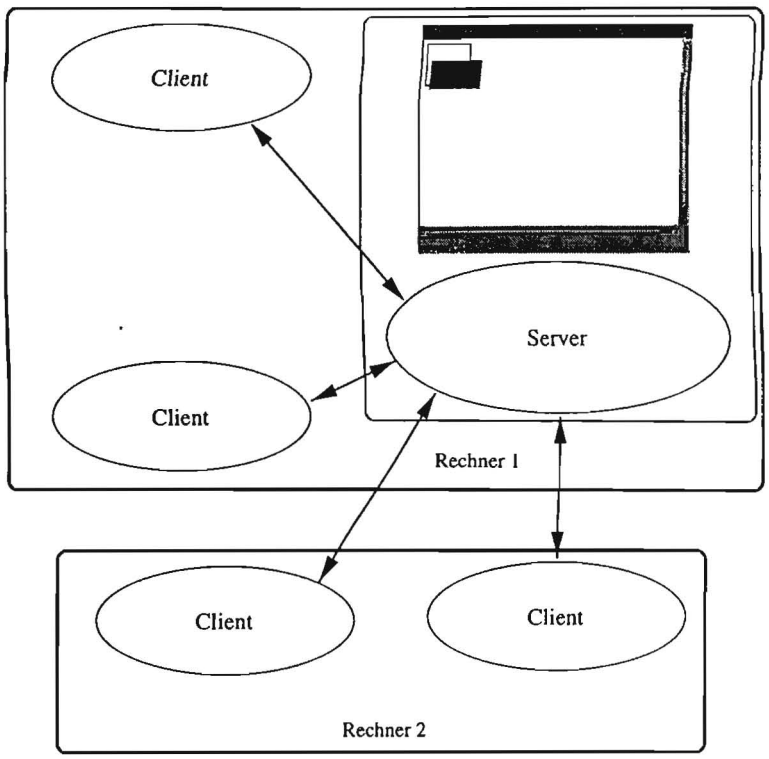

Abbildung 1: NetGLTool mit mehreren Client-Programmen

durch Konvention festgelegten Port, während für nicht-standard-Dienste festgelegte Bereiche von Port-Nummern existieren.
Der Server belegt beim Start eine solche Port-Nummer und wartet daraufhin auf Verbindungen. Der Aufbau der Verbindung wird von einem anderen Programm, dem Client, unter Angabe derselben Port-Nummer initiiert und vom Server angenommen. Das Client-Programm kann nun über die Netzwerkverbindung Befehle absenden. Der Server führt diese Befehle aus und sendet gegebenenfalls Rückmeldungen (in Form von ASCII-Texten) an das Client-Programm zurück. NetGLTool ist grundsätzlich auf den parallelen Betrieb mit mehreren Clients ausgelegt, d.h. es können mehrere Clients gleichzeitig Verbindung mit dem Server aufnehmen und Kommandos absetzen; der Server erledigt die Synchronisation der Ausgabeanforderungen zwischen den Clients selbständig.
Bei Start des Servers wird ein Fenster mit einer Zeichenfläche geöffnet, so daß ohne weitere Vorkehrungen Graphik ausgegeben werden kann. Bei Bedarf werden, durch Kommandos gesteuert, weitere Graphik-Fenster und GUI-Elemente hinzugefügt.
Der Verbindungsaufbau zum Server erfolgt, wie unter Unix üblich, über das BSD-Socket-Interface des Betriebssystems. Die einfachste Möglichkeit bietet hierbei das Programm `telnet`; es realisiert ein virtuelles Terminal, über das Kommandos an den NetGLTool-Server abgesetzt werden können. Eine ähnliche Möglichkeit bietet das Tcl-Script `netpipe`, das diesem Programmpaket beiliegt. Auch mit ihm können Kommandos an den Server abgesetzt werden, es ist jedoch mehr für die Benutzung in einer Unix-Pipe ausgelegt. Für die Benutzung aus C++-Programmen existiert letztlich die Library `libsocket++`, die Bestandteil dieses Programmpakets ist; sie realisiert eine C++-Klasse, mit der auf einfachste Weise eine Stream-orientierte Netzwerkverbindung aufgebaut werden kann.

3.1 Features

Im folgenden werden einige grundlegende Features von NetGLTool auf Ebene der Kommandosprache erläutert.

3.1.1 Koordinatensystem

Grundlage der graphischen Kommandosprache ist ein Weltkoordinatensystem, das auf Fließkommazahlen als Koordinaten beruht. NetGLTool stellt einen Ausschnitt dieses Weltkoordinatensystems (=Weltkoordinatenfenster) in einem Ausschnitt des Bildschirmfensters (=Viewport) dar (siehe Abb. 2).Bei Veränderung des Weltkoordinatenfensters ändert sich also der Anteil der Graphikausgabe, die man in einem gleichbleibendem Bildschirmausschnitt sieht; bei Änderung des Viewports ändert sich die Größe des Bildschirmausschnitts, in dem man einen gleichbleibenden Ausschnitt der Graphikausgabe sieht (Window-Viewport-Transformation). Die Größe des Bildschirmfensters kann durch das Kommando `prefsize()` gesetzt werden.

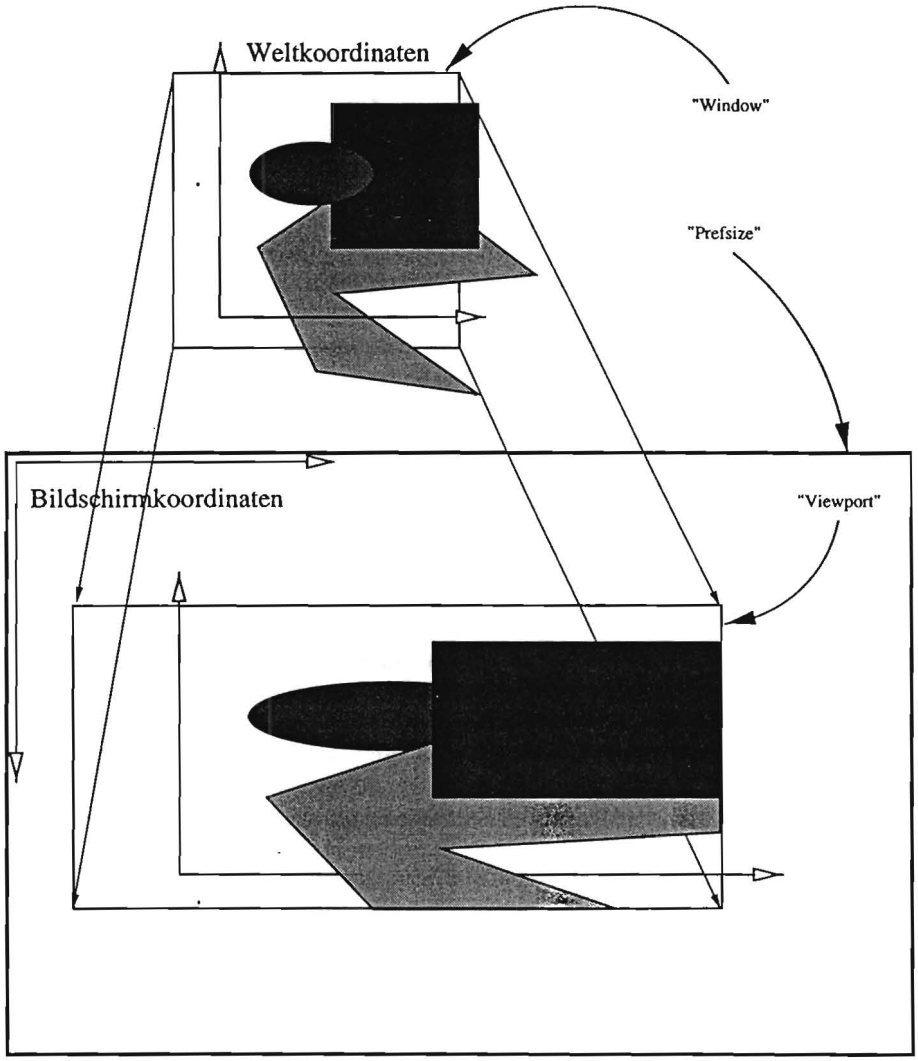

Abbildung 2: Die Window-Viewport-Transformation definiert eine Abbildung von einem Ausschnitt in Weltkoordinaten (Window) in einen Ausschnitt in Bildschirmkoordinaten (Viewport).

Zusätzlich zu dieser Window-Viewport-Transformation lassen sich lokale Koordinatensysteme einführen, indem das Weltkoordinatensystem mit einfachen Befehlen verschoben, skaliert und rotiert werden kann. Dadurch lassen sich zum Beispiel positionsunabhängige Makros definieren, die dann an jeder beliebigen Position im Weltkoordinatensystem eingesetzt werden können.

3.1.2 Graphische Primitive

Ein Großteil der GL-Kommandos hat den Zweck, graphische Ausgaben in eines der geöffneten Graphik-Fenster zu tätigen. Dabei sind grundsätzlich 3 Arten von Ausgabe-Primitiven zu

unterscheiden:

Vektorgraphik Dies umfasst einfache geometrische Primitive, nämlich Linien, Rechtecke, Polygone, Kreise. Die flächigen Primitive können dabei mit Farben und Hintergrundmustern (*stipples*) gefüllt werden.

Rastergraphik Damit sind rechteckige Farb-, Graustufen- oder Schwarzweiß-Bilder gemeint, die aus Dateien geladen und in einen Bereich des Graphik-Fensters kopiert werden. Damit `ngltool` auch mit großen und/oder vielen Bildern effizient umgehen kann, besitzt das Programm einen internen *Cache*, der den Speicher für die Bilddaten verwaltet und bei Überschreiten einer einstellbaren Speichernutzungs-Grenze Bilddaten in eine Datei auslagert. Dieser Vorgang ist für den Benutzer transparent.

Textausgabe In die Graphikfenster kann, unter Benutzung von skalierbaren X-Windows-Schriften, Text ausgegeben werden. Dabei kann eine Schrifthöhe in Weltkoordinaten angegeben werden, die Schrift wird dann mit der gegebenen Window-Viewport-Transformation skaliert.

3.1.3 Double-Buffering

Zur Effizienz-Steigerung und um Animationen zu ermöglichen, erfolgt die gesamte Graphikausgabe in einen nicht sichtbaren Hintergrund-Puffer. Mit einem besonderem Befehl (`swapbuffers()`) wird die bis dahin erfolgte Ausgabe aus diesem Hintergrund-Puffer in das Bildschirm-Fenster kopiert. Der Kopiervorgang wird unter X-Windows meistens von der Hardware unterstützt (BitBlt), so daß hierfür kein Zeitaufwand entsteht. Durch das Speichern der Ausgabe in einem Puffer sind Scroll-Operationen und Bildschirm-Updates sehr effizient durchführbar. Außerdem braucht der Fensterinhalt bei diesen Operationen nicht explizit gelöscht zu werden, was sich in einer weiteren Effizienzsteigerung und flimmerfreien Updates beim Verschieben des Fensterinhalts bemerkbar macht. Um animierte Ausgaben vor einem statischem Hintergrund zu tätigen, bietet das Programm auch die Möglichkeit, direkt in das Fenster zu zeichnen.

3.1.4 Display-Listen

Display-Listen sind ein weiteres Mittel zur Effizienz-Steigerung. Eine Display-Liste ist eine Liste von Kommandos, die "aufgezeichnet" und im Server unter einem frei vergebbarem Namen gespeichert wird. Eine solche Display-Liste kann später, unter Bezug auf diesen Namen, mit einem besonderem Befehl abgespult werden. Durch das Speichern von Kommandos auf Server-Seite kann bei wiederholter Ausführung einer Kommandofolge eine erhebliche Menge an Netzwerk-Verkehr eingespart werden. Zudem lassen sich, zusammen mit den Befehlen für die Koordinaten-Transformationen, Makro-Bibliotheken aufbauen. Die Makros können dabei aus beliebig komplexen Graphik-Objekten bestehen (z.B. auch verschachtelten Display-Listen) und in der gewünschten Position, Größe und Ausrichtung dargestellt werden.

3.1.5 Attribute-Stack

Graphikausgaben werden sehr häufig über hierarchische Modelle realisiert, d.h. ein Ausgabe-Element besteht aus einer Ansammlung von kleineren Komponenten mit gemeinsamen graphischen Attributen (z.B. Linienbreite), die wiederum aus kleineren Komponenten mit gemeinsamen Attributen bestehen, die wiederum aus Komponenten oder graphischen Primitiven bestehen. Diesem Umstand kommt das Konzept eines Attribute-Stacks entgegen. Eine Gruppe von Attributen (Farbe, Linienbreite, Linienart, Füllmuster, Schriftart und Schrifthöhe) kann mit einem Kommando gesichert und später wieder hervorgeholt werden. Nur der Top of Stack wird über die Attribute-Kommandos verändert. Die Attribute, die auf dem Top of Stack gespeichert sind, bestimmen das Aussehen der Ausgabe. Dieses Vorgehen sichert auch eine gute Effizienz, da

- an Ausgabekommandos nur Geometriedaten und keine Informationen z.B. über Farben o.ä. übergeben werden müssen (Parameter-Minimierung).

- das System große Freiheiten bei der Optimierung der Attribute-Verwaltung hat (z.B. durch Verzögern und Zusammenfassen von Änderungen).

3.1.6 Event-Behandlung, Picking

Ein Anwender-Programm will häufig Rückmeldungen über Aktionen des Benutzers erhalten. Dies geschieht bei NetGLTool in Form von ASCII-Texten; der NetGLTool-Server sendet dabei drei Arten von Rückmeldungen:

- Events als direkte Rückmeldungen von Benutzer-Aktionen (z.B. Maus-Aktionen)
- Stringlisten als Antwort auf einen `select()`-Aufruf (Picking).
- Strings als Antwort auf eine Menü-Auswahl.

Üblicherweise sieht die Benutzung so aus, daß das Anwender-Programm in einer Schleife Events von der Netzwerk-Verbindung liest und auswertet. Mit den Events erhält die Anwendung auch deren Bildschirmkoordinaten; diese können, mit Hilfe des `select()`-Befehls, zum Picken von graphischen Ausgabeelementen im Graphikfenster benutzt werden.

3.1.7 GUI-Steuerung

NetGLTool bietet ein einfach zu benutzendes Menü-System. Die Menü-Hierarchie wird durch Angabe der Pfade zu den Menü-Einträgen erzeugt, ein Pfad repräsentiert hier die Untermenüs, die durchlaufen werden müssen, um zu dem Menü-Punkt zu kommen. Die Menüeinträge werden nach Beendigung des zugehörigen Clients wieder entfernt, so daß jeder Client sein eigenes Menüsystem verwalten kann und der Zustand der Menüleiste immer den aktuellen Zustand des Gesamtsystems reflektiert.
Es existieren weiter Kommandos zur Erzeugung und Handhabung von sekundären Graphikfenstern sowie zur Ein- und Ausgabe von Text in einfachen Dialogfenstern. Weiter beinhaltet die GL-Kommandosprache Befehle zur Darstellung von gerichteten Graphen in einer Dialogbox.

3.1.8 Lupe

Die Zeichenfläche besitzt eine eingebaute Lupe, die durch Drücken der rechten Maustaste aktiviert werden kann; durch Verschieben der Maus bei gedrückter Taste kann der vergrößerte Ausschnitt verschoben werden. Vergrößerungsfaktor und Abmessungen der Lupe können über X-Resourcen eingestellt werden.

4 Bedienung von Server und Clients

In diesem Abschnitt wird die grundlegende Bedienung des Server-Programms sowie der Verbindungsaufbau mit einfachen Clients erläutert.

4.1 Erste Schritte

Zum Erlernen der Befehlssyntax durch Ausprobieren kann `ngltool` interaktiv über das Unix-Dienstprogramm `telnet` bedient werden. Dazu wird der Server einfach mit dem Kommando

```
% ngltool
```

aufgerufen. Ohne weitere Kommandozeilenparameter sucht sich der Server selbständig eine Port-Nummer und gibt diese auf der Standardausgabe aus. Die Port-Nummer kann auch mit einem Kommandozeilen-Parameter festgelegt werden:

```
% ngltool -port neue_zahl
```

Die Nummer kann beliebig, muß aber größer als 1000 sein. Bricht der Server daraufhin seine Ausführung direkt mit der Meldung

```
"bind: Address already in use"
```

ab, so heißt das, das die benutzte Internet Port-Nummer bereits von einem anderen laufenden ngltool-Server benutzt wird.
Nun kann der Server mit dem Befehl

```
telnet rechnername portnummer
```

erreicht werden, es können direkt Kommandos an den Server abgesetzt werden; Events vom Server werden auf dem Bildschirm ausgegeben. Das `telnet`-Programm kann leider keine Kommandos aus einer Datei-Umlenkung (bzw. pipe) empfangen. Für diesen Fall existiert das Tcl-Skript `netpipe` , es erfüllt im wesentlichen die gleiche Aufgabe wie telnet, ist jedoch pipe-fähig. So kann zum Beispiel die Datei `test.gl` wie folgt an den server übertragen werden:

```
netpipe rechnername portnummer < test.gl
```

Hier ist jedoch zu beachten, daß bei Dateiende auch das Skript terminiert und damit die Verbindung unterbrochen wird.

4.2 Application-Defaults

Wie alle X-Windows Programme bezieht `ngltool` eine ganze Reihe von Voreinstellungen aus sog. X-Resourcen. Diese kommen zum Teil aus einer applikationsspezifischen Datei im Installationsverzeichnis von `ngltool`, zum anderen können auch in der benutzerspezifischen Resourcen-Datei (`$HOME/.Xdefaults`) Voreinstellungen gemacht werden. Damit `ngltool` korrekt funktioniert, muß es auf alle Fälle seine applikationsspezifische Resourcen-Datei (namens `NetGLTool`) auffinden können. Im Umfeld der OMEGA-Projektgruppe ist es leider nicht ohne weiteres möglich, diese Datei im X-Windows Standard-Pfad zu installieren, daher muß eine Umgebungsvariable gesetzt werden, die das richtige Verzeichnis enthält. In der C-Shell geschieht dies mit dem Kommando

```
setenv XUSERFILESEARCHPATH /home/omega/lib/X11/%T/%N%S
```

Diese Variable besteht aus einem Pfad, unter dem Defaults-Dateien zu finden sind, jeweils mit der Endung `%T/%N%S`. Näheres hierzu findet sich in [ORA3] sowie [ORA4] .
Mit den nachfolgend beschriebenen Einstellungen kann das Verhalten von `ngltool` beeinflußt werden. Alle diese Einstellungen können wahlweise als Kommandozeilenoption (mit einem '-' eingeleitet) oder als X-Resource (durch Festlegung in der Datei `$HOME/.Xdefaults`) festgelegt werden.

port Angabe einer festen Portnummer (Voreinstellung: 0, d.h. es wird automatisch die nächste freie Portnummer gesucht).

imageCacheSize Größe des Cache für skalierte Rasterbilder, in Megabyte (Voreinstellung: 32)

cacheableScales Aufzählung von Skalierungs-Faktoren, bei denen der Cache für Rasterbilder aktiv wird (Voreinstellung: 0.1 0.25 0.5 1.0). Bei Spezifizierung auf der Kommandozeile müssen alle Faktoren zusammen in Hochkommata eingeschlossen werden. Wird hier der leere String angegeben, so ist der Cache bei allen Skalierungen aktiv. Alternativ können die Wörter `all` (Aktivierung des Cache bei allen Vergrößerungen) und `none` (Cache abgeschaltet) angegeben werden

4.3 Vorgegebene Bedienelemente

Die Oberfläche des Servers beinhaltet, neben der Zeichenfläche, eine Status-Zeile mit folgenden Bedienelementen:

- Ein Schieber (Scale) für die Einstellung des Skalierungsfaktors.
- Ein OptionMenü, das die cacheableScales enthält. Hier können die bezeichneten Skalierungen direkt ausgewählt werden.
- Ein Text-Label, das die benutzte Port-Nummer anzeigt.
- Ein LED-artiges Rechteck, das den Betriebszustand des Programms andeutet: Wenn die Fläche schwarz ist, so ist der Server inaktiv und wartet auf ankommende Kommandos. Wenn sie grün gefüllt ist, so findet Netzwerkverkehr statt und bei roter Färbung ist das Programm mit der Ausführung von Kommandos beschäftigt.

4.4 Ein interaktives Testprogramm

Im Quelltextverzeichnis (`/home/omega/src/netvisu/apps/NetGLTool/examples`) befindet sich ein Test-Client namens `ngltest`, mit dem verschiedene Features des Systems interaktiv ausprobiert werden können. Das Test-Programm wird mit `ngltest <hostname> <port>` gestartet, es kann nur aus seinem Verzeichnis heraus gestartet werden. Über das Menü kann eine Suite von Beispielen abgerufen werden, diese stehen zum Teil in Form von GL-Eingabedateien im gleichen Verzeichnis.

4.4.1 Erweiterbarkeit des Testprogrammes

Das Programm `ngltest` ist ein einfaches Tcl-Skript, das einen Unix-Socket bedient, es ist daher sehr einfach erweiterbar. Um eine weitere GL-Kommandodatei einzubinden, ist der Aufruf

```
defTestWithInputFile $socket Datei
```

im hinteren Teil der Skript-Datei einzufügen. Um eine interaktive Funktionalität aufzunehmen, ist ein Aufruf von

```
defTestWithProc $socket Prozedurname
```

einzufügen, `Prozedurname` taucht dann als Menüpunkt auf und wird, mit dem Socket als Argument, bei Auswahl des Menüpunktes aufgerufen. Die Prozedur kann dann über den Socket Kommandos absetzen und Events empfangen. Nähere Informationen können dem Skript-Text entnommen werden.

5 GL-Befehlsreferenz

Die GL-Befehlssprache ist grundsätzlich zeilenorientiert, d.h. jeder Befehl wird durch ein Zeilenende-Zeichen (ASCII 13) abgeschlossen. Die meisten Befehle (genauer gesagt, Befehle für die Graphik-Ausgabe) werden nicht direkt ausgeführt, sondern gepuffert und zu definierten Zeitpunkten (bei Auftreten bestimmter Befehle) ausgeführt. Zustandsändernde Befehle (im besonderen Befehle zur Manipulation von GUI und gerichteten Graphen) werden dagegen unmittelbar ausgeführt und nicht gepuffert. Für Testzwecke ist die Verwendung eines Kommentarzeichens ("#") erlaubt, dieses Zeichen muß als erstes Zeichen der Zeile erscheinen (außer Whitespaces), der Rest der Zeile wird ignoriert.

5.1 Datentypen

Der Parser von NetGLTool erkennt folgende Datentypen:

Zeichenketten (string) Strings sind eine Folge von ASCII-Zeichen, eingeschlossen in Double-Quotes, z.B. `"xyz"`. Die Double-Quotes sind optional, wenn der String nur alphanumerische Zeichen enthält. Backslash-Codes für Sonderzeichen, wie sie in C verwendet werden, sind erlaubt (Nur in gequoteten Strings!). Falls im String selber Double-Quotes vorkommen sollen, so muß ein Backslash vorangestellt werden (wie in C).

Fließkomma-Zahlen (float) GL verwendet Fließkomma-Zahlen doppelter Genauigkeit, sie entsprechen dem Typ `double` in C und haben die gleiche Schreibweise.

Integer-Zahlen (int) Vorzeichenbehaftete, ganze Dezimalzahlen mit 32 Bit Genauigkeit.

Punkte (point) zwei Fließkommazahlen, eingeschlossen in Klammern und durch ein Komma getrennt (z.B. `(10.0,25.0)`).

5.2 Sichtdefinition

window(float x, float y, float w, float h) legt den Ausschnitt des Weltkoordinatensystems fest, der im Viewport dargestellt wird. Die linke obere Ecke des Weltkoordinatenfensters liegt bei *(x,y)* und hat die Breite w sowie die Höhe h, jeweils in Weltkoordinaten. Durch Angabe einer negativen Breite bzw. Höhe läßt sich ein seitenverkehrtes Koordinatensystem definieren.

viewport(int x, int y, int w, int h) legt den Ausschnitt des Bildschirmfensters fest, in dem das Weltkoordinatenfenster dargestellt wird. Die obere linke Ecke *(x,y)*und die Breite w bzw. Höhe h werden in Bildschirmkoordinaten spezifiziert.

5.3 Transformationen, der Matrix-Stack

Zusätzlich zu der Sichtdefinition können die Ausgabeprimitive einer Koordinaten-Transformation unterzogen werden. Durch diese Möglichkeit kann jedes Ausgabe-Objekt verschoben, skaliert und rotiert werden. Alle Transformationen werden akkumuliert, das heißt, daß die Wirkung einer Transformation von den vorhergehenden abhängt. Dabei ist zu beachten daß diese Operationen im allgemeinen nicht kommutativ sind.
Um nun mit diesem Konzept lokale Koordinatensysteme einführen zu können, benötigt man eine Möglichkeit, Transformationen wieder zurückzunehmen. Das heißt, wenn man z.B. eine Rotation ausführt, ein Objekt zeichnet und dann eine Translation durchführt, noch ein Objekt zeichnet und dann mit dem Zeichnen des ersten Objekts weitermachen will (und zwar in dessen Koordinatensystem), muß man die Translation wieder zurücknehmen. In NetGLTool wird das so realisiert, das alle lokalen Koordinatensysteme auf einem *Stapel* organisiert werden. Die Semantik diese Stapels ist dabei folgende:

- Die aktuell eingestellte Transformation befindet sich auf dem Top Of Stack und wird zur Ausführung der nachfolgenden Ausgabe-Befehle benutzt. Alle Befehle zur Änderung der Transformation wirken sich auf diesen Top of Stack aus.
- Bei einer *Push*-Operation wird der Top of Stack dupliziert. Der neue Top of Stack kann nun durch `translate`, `scale` und `rotate` manipuliert werden, ohne daß sich dies auf die alte Transformation auswirkt. Nun können Ausgaben in diesem neuem Koordinatensystem getätigt werden.
- Bei einer *Pop*-Operation wird das alte Koordinatensystem wiederhergestellt.

Auf diese Weise ist es möglich, beliebig komplexe Strukturen hierarchisch ineinander zu verschachteln und aus Grundprimitiven aufzubauen. Dabei kann insbesondere das Konzept der Display-Listen nützlich sein, die eine Art der Unterprogramm-Technik ermöglichen.

pushmatrix() Top of Stack des Matrix-Stack kopieren und Kopie zum neuen Top of Stack machen.

popmatrix() Top of Stack vom Matrix-Stack entfernen. Zur Beachtung: der Matrix-Stack darf nie leer werden! Ein Versuch, das letzte Element vom Stack zu nehmen, wird mit einer Fehler-Meldung quittiert und nicht ausgeführt.

loadidentity() Top of Stack mit der Identitäts-Matrix initialisieren. Das lokale Koordinatensystem ist damit identisch zum Welt-Koordinatensystem.

translate(double x, double y) Top of Stack mit einer Translations-Matrix multiplizieren. Der Ursprung des aktuellen Koordinaten-Systems wird an den Punkt *(x, y)* verschoben.

scale(double x, double y) Top of Stack mit einer Skalierungs-Matrix multiplizieren. Das aktuelle Koordinaten-System wird in x/y-Richtung um die entsprechenden Faktoren gedehnt.

rotate(double phi) Top of Stack mit einer Rotations-Matrix multiplizieren. Die Rotation erfolgt um den Punkt *(0,0)* des lokalen Koordinatensystems.

rotate(double x, double y, double phi) Top of Stack mit einer Rotations-Matrix multiplizieren. Die Rotation erfolgt um den Punkt *(x,y)* des lokalen Koordinatensystems.

5.4 Attribute

Zum Zeichnen werden von allen Routinen grundsätzlich die Attribute des TOS (Top of Stack) des Attribute-Stacks verwendet. Alle Befehle zum Setzen der Attribute wirken sich auf den TOS aus.

linewidth(int w) Setze Linienbreite für Linien, Rechtecke und Polygone. VORSICHT: die Linienbreite bezieht sich auf den Graphik-Kontext von X-Windows und wird NICHT skaliert! (siehe XSetLineAttributes [ORA2]).

linestyle(int style) Setze X11 Linientyp: 0 = LineSolid, 1 = LineOnOffDash, 2 = LineDoubleDash (siehe XSetLineAttributes [ORA2]).

color(float r, float g, float b) Setze aktuelle Farbe für alle Zeichenbefehle. Die Farbwerte (r,g,b) sind normalisiert (im Bereich *[0.0 .. 1.0]*) und werden aus einer Standard-Farbpalette angenähert.

pushattributes() Dupliziere TOS des Attribute-Stacks.

popattributes() Entferne den TOS des Attribute-Stacks.

defstipple(name, width, height, byte_1, byte_2, ..., byte_n) Definiere Füllmuster. Es wird eine monochrome Bitmap mit der Breite *width* und der Höhe *height* definiert und unter dem Namen *name* abgespeichert. Die Bytes der Bitmap werden als Argumente *byte_1* bis *byte_n* spezifiziert. Die Bitmap kann später mit dem Befehl `setstipple()` unter dem angegebenem Namen referenziert werden.

setstipple(string name) Setze das Füllmuster für Text, Linien und gefüllte Flächen. Der Parameter *name* ist ein Bezeichner, unter dem mit `defstipple()` ein Füllmuster definiert wurde. Der vordefinierte Bezeichner `None` ist dafür vorgesehen, die Benutzung eines Stipples wieder abzuschalten.

5.5 Rastergraphik

NetGLTool bietet die Möglichkeit, Rasterbilder von Datei zu Laden und unter einem Namen abzuspeichern. Beim Darstellen im Graphikfenster wird es der Window-Viewport-Transformation entsprechend skaliert.

loadimage(string filename, string id) lade die Bilddatei *filename* und lege das Bild unter dem Namen *id* ab. ZUR BEACHTUNG: *filename* muß ein absoluter Dateiname auf dem Rechner sein, auf dem der NetGLTool-Server läuft, da die Bilddatei sonst nicht gefunden werden kann. Falls der Name `id` bereits für ein Bild verwendet wird, so wird das alte Bild gelöscht und das neue Bild unter dem Namen abgespeichert.

deleteimage(string id) lösche Bild *id*. Das Bild kann nach Ausführung dieses Befehls nicht mehr referenziert werden.

putpixels(string id, float x, float y, float w, float h) kopiere das Bild *id* in den durch das Rechteck *(x,y,w,h)* (linke obere Ecke, Breite, Höhe) gegebenen Weltkoordinatenbereich. Das Bild wird, wie alle Graphik-Primitive, entsprechend der Window-Viewport-Transformation skaliert, jedoch findet (bisher noch nicht) eine *Rotation* des Bildes statt, stattdessen wird nur die linke obere Ecke rotiert.

5.6 Vektorgraphik

Alle Vektorgraphik-Befehle beziehen sich auf das Weltkoordinatensystem. Skalierungen und Verschiebungen ergeben sich aus der Window-Viewport Transformation. Alle Vektorgraphik-Befehle benutzen den Top of Stack des Attribute-Stacks für die Spezifikation von Linienbreite, Linientyp, Farben, Füllmuster und ggf. der Schriftart.

drawline(float x1, float y1, float x2, float y2) Zeichne eine Linie von *(x1,y1)* nach *(x2,y2)*.

rect(float x, float y, float w, float h) Zeichne ein horizontales, nicht gefülltes Rechteck am gegebenen Punkt *(x,y)* und mit den Abmessungen *(w,h)*.

rectf(float x, float y, float w, float h) Zeichne ein horizontales, gefülltes Rechteck am gegebenen Punkt *(x,y)* und mit den Abmessungen *(w,h)*.

arc(float x, float y, float r, float h, float s, float e) Kreisbogen zeichnen. Mittelpunkt ist *(x,y)*, die Größen *r,h* geben den horizontalen bzw. vertikalen Radius an. *s* ist der Winkel, von der Waagrechten aus gemessen, bei dem der Kreisbogen beginnt, *e* die Ausdehnung des Bogens. Beide Winkel werden in Grad angegeben.

arcf(float x, float y, float r, float h, float s, float e) Gefüllten Kreisbogen ("Tortenstück") zeichnen. Parameter wie bei `arc()`.

poly(point P1, point P2, ..., point Pn) Zeichne ein nicht gefülltes Polygon mit den Eckpunkten *P1...Pn*.

polyf(point P1, point P2, ..., point Pn) Zeichne ein gefülltes Polygon mit den Eckpunkten *P1...Pn*.

bgnpoly(), endpoly() Beginne/Beende Punkteliste für ein nicht gefülltes Polygon.

bgnpolyf(), endpolyf() Beginne/Beende Punkteliste für ein gefülltes Polygon.

vertex(float x, float y) Nehme Polygonpunkt zur Punkteliste eines Polygons hinzu.

Beispiel für ein Polygon in der bgnpoly()/endpoly()-Schreibweise:

```
bgnpoly()
  vertex(10,10)
  vertex(20,10)
  vertex(20,20)
  vertex(10,20)
endpoly()
```

Sinn und Zweck dieser Schreibweise ist es, bei sehr komplexen Polygonen (z.B. über 100 Punkte) die Zeilenlänge nicht ausufern zu lassen.

5.7 Text, Schriftarten

Zur Ausgabe von Text werden X-Windows Schriftarten benutzt, Ausgangspunkt für deren Festlegung ist die XLFD, X Logical Font Description (Näheres siehe [ORA1]). In GL werden Schriftgrößen in Weltkoordinaten festgelegt, d.h. bei Setzen einer Skalierung über die `window()` / `viewport()`-Kommandos werden auch die Schriften skaliert. Damit das funktioniert, sind ausschließlich skalierbare Fonts zulässig. In der XLFD hat eine Schrift-Spezifikation folgendes Aussehen:

```
-adobe-courier-bold-o-normal--11-80-100-100-m-60-iso8859-1
```

Die erste Hälfte, bis zum "--" des Namens, wird hier vereinfacht als *fontbase* bezeichnet und kann z.B. mit dem Kommando `setfontbase("adobe-courier-bold-o-normal")` gesetzt werden. Im obigen Beispiel ist das die Courier-Schriftart von Adobe, fett gedruckt (bold) und kursiv (o = Oblique). Die zweite Hälfte spezifiziert Dimensionen und Auflösung der Schrift und wird von `ngltool` vervollständigt. X-Windows-Fonts lassen sich mit dem Unix/X11-Befehl `xlsfonts` auflisten, die skalierbaren Fonts findet man durch folgenden Aufruf:

```
xlsfonts -fn "*--0-0-*"
```

Die Dimensionierung "*–0-0-*ßpezifiziert eine skalierbare Schriftart. Fonts werden, wie alle anderen Attribute auch, auf dem Attribute-Stack verwaltet.

setfontbase(string name) Setze Font-Prefix für aktuelle Schriftart. Die Schriftart, die bei Start von `ngltool` gültig ist, läßt sich über die Resource `defaultFontHeight` der Zeichenfläche einstellen, z.B. durch Angabe von

```
NetGLTool*defaultFontBase: adobe-courier-medium-r-normal
```

in der Datei `$HOME/.Xdefaults`.

setfontheight(float h) Setze Schrifthöhe in Weltkoordinaten. Die Font-Höhe, die bei Start von `ngltool` gültig ist, läßt sich über die Resource `defaultFontHeight` der Zeichenfläche einstellen, z.B. durch Angabe von `NetGLTool*defaultFontHeight: 28` in der Datei `$HOME/.Xdefaults`.

drawstring(string s, double x, double y) Zeichne die Zeichenkette *s* an die Position *(x,y)*.

textwidth(string s) Abfrage der Breite eines Textes unter den aktuellen Font-Attributen und Transformation. Die Breite wird als Detail eines 'Textwidth'-Events zurückgeliefert. BEACHTE: Der tatsächlich von `ngltool` benutzte Font hängt von der Transformation und damit auch vom Zoom-Faktor ab, der von `textwidth()` zurückgelieferte Wert kann daher (trotz Normalisierung auf Weltkoordinaten) für unterschiedlich eingestellte Zoom-Faktoren um einen Bruchteil variieren.

5.8 Display-Listen

Display-Listen sind Verkettungen von GL-Kommandos, die im NetGLTool-Server unter einem Namen abgespeichert und zu einem späteren Zeitpunkt ausgeführt werden können. Dieses Feature kann man als Makro- oder Unterprogramm-Technik auffassen. Dabei ist zu beachten, daß die Listen-Kommandos zum *Definitions-Zeitpunkt* (d.h. innerhalb der Befehle `newlist()`, `endlist()`) keinen Effekt haben.

newlist(string name) Beginne das Aufzeichnen von Befehlen; es wird eine Display-Liste mit Namen `name` erzeugt. Ist eine solche Display-Liste bereits vorhanden, so wird diese überschrieben. Alle nachfolgenden Befehle bis zum nächsten `endlist()` werden in diese Display-Liste eingefügt und können dann mit `calllist()` abgespult werden. Die aufgezeichneten Kommandos werden während des Aufzeichnens NICHT ausgeführt, d.h. um deren Effekt zu erzielen, MUSS die Liste ausgeführt werden.

endlist() Beende das Aufzeichnen von Befehlen (Gegenstück zu `newlist()`)

appendlist(string name) Erweitere Display-Liste mit Namen `name` um neue GL-Kommandos. Es wird, wie bei `newlist`, eine Liste von Kommandos aufgezeichnet, die mit `endlist()` abgeschlossen wird. Die neu erzeugte Liste wird an die bestehende angehängt.

calllist(string name) Führe Display-Liste `name` aus. Alle in der Display-Liste gespeicherten Kommandos werden nacheinander ausgeführt. Beachte: dieser Befehl stellt einen *Zeichen-Befehl* dar, wird also gepuffert, bis ein `swapbuffers()` oder `flush()` die Ausgabe sichtbar macht.

5.9 NameStack, Picking

Der NameStack ist ein Stack aus Strings, der mit GL-Befehlen manipuliert werden kann. Auf Anforderung kann NetGLTool den NameStack-Inhalt, wie er während des Zeichnens der graphischen Elemente auf einer spezifizierten Bildschirmkoordinate bestand, zurückgeben (`selectlist()`-Befehl). Das Konzept kommt dem Umstand entgegen, daß Graphik-Ausgaben meistens in einer hierarchisch strukturierten Art und Weise erzeugt werden, der Stack-Inhalt gibt hierbei während des Zeichnens die Hierarchie wieder. Ein Vorteil gegenüber einem rein objektorientierten Konzept ist, daß der Anwender freie Wahl bei der Zuordnung von Hierarchieebenen zu graphischen Elementen hat, er kann Kommandos zur Manipulation des Namestack frei zwischen die eigentlichen Ausgabe-Kommandos einstreuen. Ein weiterer Vorteil ist, daß zur Erzeugung der Hierarchie keine Objekte übergeben werden müssen, sondern nur Ausgabe getätigt wird. Zum besseren Verständnis des Konzepts sei hier die Implementierung angedeutet: der Server `ngltool` speichert intern alle Kommandos, die zum aktuellen Bildschirm-Inhalt geführt haben, in einer Display-Liste. Auf einen `selectlist()`-Befehl hin wird diese Display-Liste ausgeführt; jedoch werden die Graphik-Kommandos nicht in Prozedur-Aufrufe zur graphischen Ausgabe umgesetzt, sondern es wird zu jedem Kommando geprüft, ob die Bildschirm-Koordinate des `selectlist()`-Aufrufs innerhalb der graphischen Ausgabe des Kommandos liegt. Ist dies der Fall, so wird der aktuelle Inhalt des NameStacks (der ja durch die NameStack-Kommandos in der Display-Liste manipuliert wurde) beim nächsten `popname()` oder `loadname()`-Aufruf in die Event-Queue des Servers eingereiht und das Client-Programm erhält den NameStack-Inhalt in Form einer String-Liste. Dabei können pro `selectlist()`-Aufruf mehrere Zeilen zurückgegeben werden (eine pro graphischem Element, das "getroffen" wurde).
In der Abbildung 3 sieht man, wie hierarchische Objekte ausgegeben werden können. Wird nach dem Aufbau der Zeichnung ein `selectlist()`-Kommando mit einem Koordinatenpaar aufgerufen, das innerhalb des kleinen, linken, oberen Rechtecks liegt, so würde `ngltool` die Liste `Up Left Box` zurückliefern. Der zugehörige Code zum Aufbau der Zeichnung würde folgendermassen aussehen (Die Kommentare beziehen sich auf den NameStack):

```
pushname("Box")
# aeusseren Umriss
rect(10,10,300,50)

# laengliches Kreuz
pushname("Cross")
rectf(...)
rectf(...)
popname()

# Abbildung a)

# linkes, hochkant stehendes Rechteck
pushname("Left")
color(0.0,1.0,0.0)
rectf(...)
# zwei kleine liegende Rechtecke
pushname("up")
color(1.0,0.0,0.0)
rectf(...)

# Abbildung b)

# "up" durch "Down" ersetzen
loadname("Down")
rectf(...)

# Abbildung c)
```

```
# "Down" entfernen
popname()

# Rechte Seite, zuerst "Left" durch "Right" ersetzen
loadname("Right")
color(0.0,1.0,0.0)
rectf(...)
pushname("up")
color(1.0,0.0,0.0)
rectf(...)
# "up" durch "Down" ersetzen
loadname("Down")
rectf(...)

# Abbildung d)

# "Down" entfernen
popname()
# "Right" entfernen
popname()
# "Box" entfernen
popname()
```

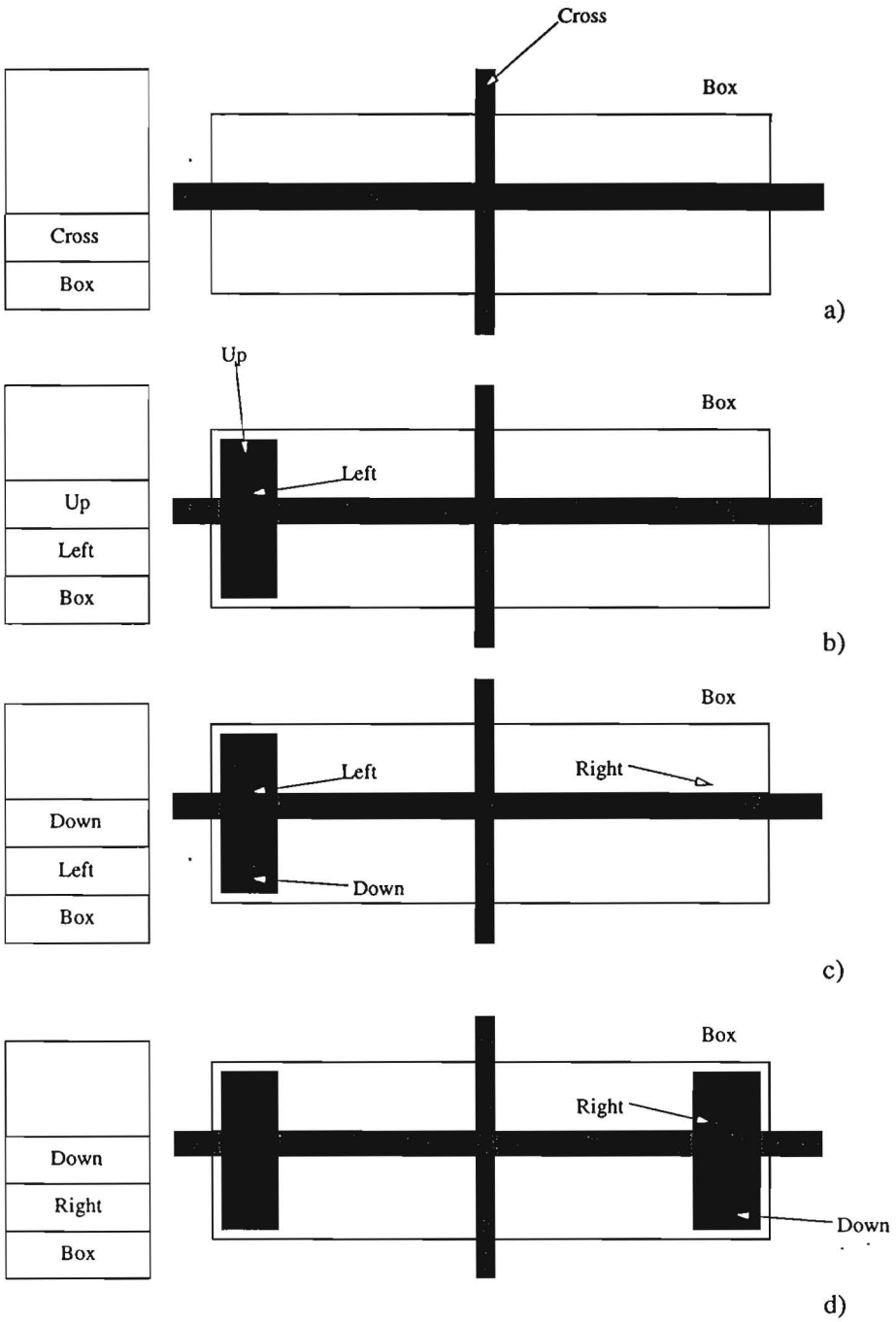

Abbildung 3: Namestack während des Zeichnens eines hierarchischen Objekts

pushname(string name) Sichere Zeichenkette *name* auf dem NameStack.

loadname(string name) Ersetze den Top of Stack des NameStacks durch *name.*

popname() Entferne den Top of Stack des NameStacks.

selectlist(float x, float y) Picking-Befehl. NetGLTool gibt den NameStack-Inhalt zurück, wie er beim Zeichnen der unter der Weltkoordinate *(x, y)* liegenden graphischen Elemente bestand. Der NameStack-Inhalt wird als Text-Zeile an den Client gesendet. Dabei können mehrere Zeilen als Folge eines select-Aufrufs (eine Zeile pro graphischem Element) an den Client zurückgeschickt werden.

select(float x, float y) Alternativer Picking-Mechanismus. Es wird nur der Top Of Stack des letzten Treffers zurückgeliefert.

screen2world(int x, int y) liefert zu einer gegebenen Bildschirmkoordinate die korrespondierende Weltkoordinate. Die Weltkoordinate wird als `Coordinate`-Event (siehe Abschnitt 6.1) zurrückgeschickt.

5.10 Steuer/GUI-Kommandos

Mit den Steuer/GUI-Kommandos können Dialoge, Menüs und sekundäre Graphikfenster gehandhabt werden. Viele Befehle beziehen sich auf Fenster-Id's, dies sind textuelle Bezeichner, unter denen alle Interface-Elemente in einer internen Tabelle registriert werden. Hier ist zu beachten, daß jeder Bezeichner nur für jeweils ein sichtbares Fenster zu verwenden ist. So ist es zum Beispiel nicht möglich, eine Text-Eingabebox zu öffnen (`gettext()`) und direkt darauf eine Text-Ausgabebox mit dem gleichen Namen. Jedoch könnte man die Ausgabebox öffnen, nachdem man die Eingabebox geschlossen hat (z.B. mit dem "OK"-Button). In der Praxis empfiehlt es sich, die Namen in irgendeiner Art Hierarchie zu strukturieren (z.B. `gettext("variable.eingabe",...)` , `textbox("variable.ausgabe",...)`.

prefsize(int w, int h) Breite/Höhe des Graphikfensters setzen.

addpanel(string id) Neues Fenster mit der geg. id erzeugen. Es wird eine komplette Instanz der Oberfläche mit Menubar und Graphikfenster ("Panel") erzeugt. *id* kann ein beliebiger Textstring sein.

settitle(string text) Fenstertitel des aktuellen Panels setzen.

makecurrent(string id) Fenster mit der geg. id als aktuelle Zeichenfläche einstellen. Alle nachfolgenden Graphikkommandos beziehen sich auf dieses Fenster.

textbox(string id, int columns, int rows) Textbox mit der gegebenen *id* erzeugen und in den Vordergrund bringen. Falls eine Textbox mit der gegebenen *id* existiert, wird die bestehende nur nach vorne gebracht. Eine Textbox ist ein einfacher Dialog mit einem Text-Ausgabeteil und einem "OK"-Button, der die Textbox wieder schließt. Die Größe des Text-Ausgabeteils beträgt *rows* Zeilen mal *columns* Zeichen. Mit dem Befehl `wputs()` kann Text ausgegeben werden.

gettext(string id, string label, int columns, int rows) Ein einfacher Text-Eingabedialog wird erzeugt. Der Dialog besteht aus einem einfachem Textlabel, einem Text-Eingabefeld und einem "OK"-Button. Der Bezeichner *id* ist die Fenster-Id für den Dialog, *label* ist der Textstring der in dem Textlabel angezeigt werden soll und *width*, *height* dimensionieren das Eingabefeld. Im Fall von $height \neq 1$ werden Scrollbalken an das Eingabefeld angefügt, ansonsten wird ein einzeiliges Eingabefeld ohne Scrollbalken erzeugt. Bei Betätigung des "OK"-Buttons wird ein GetText-Event mit dem eingegebenem Text an den Client gesendet, der "Cancel"-Button veranlaßt das Zurücksenden eines GetText-Event mit leerem Text.

addmenupath(string path, string type, int state) Menü-Punkt hinzufügen. Das Argument *path* legt die Untermenü-Punkte fest, die zum zu erzeugenden Menüpunkt führen; er wird wie ein Unix-Dateipfad zusammengesetzt, mit "/" (Slash) als Trennzeichen zwischen den Komponenten. Die letzte (oder gegf. einzige) Komponente legt die Beschriftung des Menüeintrags fest. Die bis dahin noch nicht erzeugten Menüs / Untermenüs werden ergänzt. Menüpunkte sind an den Client gebunden, der sie erzeugt, d.h. bei Abbruch

der Verbindung zu einem Client werden die von diesem Client erzeugten Menüeinträge entfernt.

Der optionale Parameter *type* kann die Werte "normal", "radio", "check" annehmen und dient dazu, Check- bzw. Radio-Boxen zu realisieren. Für die Typen "radio" und "check" bestimmt der optionale Parameter *state*, ob der Eintrag als "gesetzt" (*state*=1) oder "nicht gesetzt" (*state*=0) zu markieren ist.

delmenu(string path) Der durch *path* spezifizierte Menü-Punkt wird gelöscht. Ist *path* ein Untermenü, so wird es komplett gelöscht.

wputs(string textbox, string text) Zeichenkette *text* in Textfenster *textbox* schreiben.

wclear(string textbox) Löscht den Text in Textfenster *textbox* .

quit() beendet NetGLTool. Alle Client-Verbindungen werden daraufhin unbrauchbar und müssen geschlossen werden.

sleep(int seconds) Die Ausführung des NetGLTool-Servers wird für *seconds* Sekunden suspendiert.

5.11 Graphikpuffer, Refreshlisten

Mit den folgenden Kommandos kann das Doublebuffering beinflußt werden. Normalerweise (als Voreinstellung) beziehen sich alle Graphikkommandos auf einen Hintergrundpuffer (backbuffer), der mit dem Kommando `swapbuffers()` in das sichtbare Fenster (frontbuffer) kopiert werden kann. Mit dem Kommando `frontbuffer()` kann das Bildschirmfenster direkt als Ausgabeziel eingestellt werden. Zudem werden alle Graphikkommandos von NetGLTool in zwei internen Refresh-Display-Listen gepuffert (jeweils für den Inhalt des Backbuffer und Frontbuffer), um bei einer Änderung des Zoom-Faktors den Bildschirminhalt wiederherstellen zu können. Diese Listen können explizit geleert werden, um eine animierte Ausgabe zu vereinfachen, für eine Anwendung siehe Beispiel 7.6.

backbuffer() Schaltet den Hintergrund-Puffer als Ziel für die Graphik-Ausgabe ein. Um die nachfolgenden Ausgaben sichtbar zu machen, muß `swapbuffers()` aufgerufen werden.

frontbuffer() Schaltet das Graphikfenster als Ziel für die Graphik-Ausgabe ein. Nach einem Aufruf von `flush()` werden alle in den Frontbuffer getätigten Ausgaben direkt sichtbar (ohne `swapbuffers()`)

swapbuffers() Graphik-Ausgabe sichtbar machen. Der Hintergrund-Puffer wird in das Fenster kopiert. Dieser Befehl betrifft nicht die GUI-Kommandos (Diese werden immer direkt ausgeführt!).

flush() Graphikkommandos aus der internen Queue des Servers entfernen und direkt ausführen.

clear() Graphikausgabe löschen. Es wird der mit `frontbuffer()`, `backbuffer()` eingestellte Puffer sowie die zugehörige Displayliste gelöscht.

clearlist() Refresh-Displayliste für den eingestellten Puffer löschen. Der zugehörige Puffer wird nicht gelöscht.

5.12 Graphen

Mit den folgenden Kommandos können gerichtete Graphen dargestellt werden. Der Graph wird als Datenstruktur unter einem Namen im Server abgespeichert; es können dann ein oder mehrere Ansichten (Views) auf diesen Graph erzeugt werden. Es sind dabei 3 verschiedene Arten von Views möglich, die unterschiedliche Funktionalitäten erlauben:

1. Darstellung des Graphen in einer Dialog-Box.
2. Darstellung des Graphen durch ein externes Programm.
3. Ausgabe des Graphen in ein Graphik-Fenster.

Eine View wird, wie alle anderen GL-Objekte auch, unter einem Namen referenziert und definiert im wesentlichen Eigenschaften des Graph-Layoutings. Die Ausgabe in ein Graphik-Fenster

unterscheidet sich von den beiden anderen Ausgabe-Arten dadurch, daß ein zusätzliches Kommando ausgeführt werden muß (`drawgraph`), um den Graph tatsächlich sichtbar zu machen. Dies wird dadurch bedingt, daß in diesem Fall das Ausgabekommando in Display-Listen eingefügt und bei jedem Bildschirm-Refresh ausgeführt wird, während bei dem Graph-Dialog und dem externen Tool der Refresh von dem Fenster selbst durchgeführt wird.

newgraph(string name) Erzeugt einen neuen, leeren Graph. Die Datenstruktur kann später unter dem Namen *name* referenziert werden.

addnode(string graph, string label, string id, string type) Erzeugt im angegebenen Graphen einen Knoten. Der optionale Parameter *type* spezifiziert, ob *label* als Knoten-Beschriftung (Wert "text", Defaultwert) oder Name einer Display-Liste (Wert "list") zu interpretieren ist. Im ersten Fall wird *label* als rechteckig umrandeter Text ausgegeben, ansonsten wird die genannte Display-Liste an der Knoten-Koordinate ausgegeben. Der Knoten kann unter dem Namen *id* zur Erzeugung von Kanten referenziert werden.

addedge(string graph, string fromid, string toid, string label) Erzeugt im angegebenem Graphen eine Kante vom Knoten *fromid* zum Knoten *toid.* Diese Parameter sind die Knoten-Namen, die im Befehl `addnode()` als Parameter *id* spezifiziert wurden. In den Views, die dies erlauben (Extern und Draw), kann die Kante mit dem angegebenem *label* beschriftet werden; dieser Parameter ist optional.

bgngraph(string graph),endgraph() Anfang / Ende von Kommandos zur Graph-Erzeugung markieren. Das Update der Ansichten wird nach Aufruf von `bgngraph()` hinausgezögert, bis ein `endgraph()`-Aufruf getätigt wird. Dies erhöht die Performance des Graph-Zeichnens beträchtlich. Diese Klammerung darf nicht geschachtelt werden.

graphview(string graph,string id,string type,string gravity) Erzeuge eine neue Ansicht auf den Graph *graph.* Der Parameter *type* kann eins der Wörter `dialog`, `extern` und `draw` mit folgenden Bedeutungen sein:

`dialog` Erzeuge Dialog-Box mit Motif-Buttons als Knoten und einer graphischen Darstellung der Kanten. Die Fenster-Id des Dialogs wird auf *id* gesetzt. Das Erscheinungsbild der Knoten und Kanten (Farben, Liniendicke) kann über Resourcen eingestellt werden . Die Dialog-Box erlaubt KEINE Darstellung von Kanten-Labels. Betätigen der Knoten-Buttons liefert ein GraphNode-Event an den Client. Die Dialogbox erscheint direkt nach Ausführung des `graphview()`-Kommandos. Manipulationen des Graphen (`addnode()`, `delnode()` ...) wirken sich sofort auf die graphische Darstellung aus, da das Refresh der Dialog-Box automatisch durchgeführt wird.

`extern` Es wird ein externes Programm (Graphlet) zur Visualisierung des Graphen gestartet. Bei jeder Änderung des Graphen wird eine neue Instanz des Programms gestartet. Graphlet stellt Kanten-Labels dar, kann aber keine Events an den Client zurückliefern. Das externe Programm wird direkt nach Ausführung des `graphview()`-Kommandos gestartet. Jedes Update des Graphen hat einen erneuten Start des externen Programms mit dem modifizierten Graph zur Folge.

`draw` Der Graph kann auf der aktuellen Zeichenfläche so dargestellt werden, daß er den eingestellten Viewport ausfüllt. Um die eigentliche Ausgabe zu veranlassen, muß ein `drawgraph()`-Kommando ausgeführt werden.

Der optionale Parameter *gravity* gibt die Richtung an, in der der Graph gezeichnet wird:

WestGravity horizontal von links nach rechts

EastGravity horizontal von rechts nach links

NorthGravity vertikal von unten nach oben

SouthGravity vertikal von oben nach unten

Voreinstellung ist `WestGravity`. Dieser Parameter wird nur von der "dialog"-View beachtet.

drawgraph(string viewid) Graph-Ansicht auf die aktuelle Zeichenfläche ausgeben. Der Graph wird normalisiert ausgegeben, d.h. im Rechteck *(0.0,0.0,1.0,1.0).* Die Window/Viewport-Transformation sollte entsprechend angepaßt werden, um den Graph im gewünschtem Bildschirmbereich auszugeben.

graphbox(string name) Kombination aus `newgraph(name)` und `graphview(name,name,dialog,WestGravity)`.

gclear(string graph) löscht den angebenen Graph. Die Datenstruktur existiert danach noch, ist aber leer.

6 Rückmeldungen, Events

6.1 Allgemeines

Die Rückmeldungen, die ngltool an die Clients sendet, haben eine vereinheitlichte Form. Es gibt dabei drei Formen von Rückmeldungen:

- von X-Windows generierte Events.
- Antworten auf Anfragen durch das Client-Programm, dies sind unten aufgeführte GL-Befehle.
- Fehlermeldungen.

Alle diese Events haben die Form *Window-ID'/'Event-Art'/'Event-Detail.* Dabei sind

Window-ID Die ID des Fensters, in dem das XEvent passierte oder das das aktuelle Panel war, als der Client die Anfrage absetzte. Dies ist die ID, die bei Aufruf der Befehle `addpanel()` oder `graphbox()` angegeben wurde. (Anmerkung: Das Graphikfenster, das beim Start von ngltool erzeugt wird, hat die ID `main`).

Event-Art Ein Schlüsselwort, aus dem sich die Art der Rückmeldung ergibt, siehe unten.

Event-Detail weitere Daten zu dem Event, diese hängen von der Event-Art ab. Das Detail eines Events wird immer als Liste von Wörtern, durch Leerzeichen getrennt, an den Client übergeben. Koordinaten werden grundsätzlich als Weltkoordinaten übergeben.

6.2 Event-Referenz

Im folgenden werden die Event-Arten mit den zugehörigen Details beschrieben.

KeyPress, KeyRelease Tastatur-Event. Event-Detail ist der X-Keysym-Name der Taste. Diese Namen finden sich als C-Definitionen in der Datei `/opt/X11/include/X11/keysymdef.h` (der Prefix `XK_` wird hier weggelassen) und können mit dem X-Programm `xev` ausprobiert werden. An den eigentlichen Tastennamen wird evtl. noch eine Liste von Modifiern (Shift, Ctrl, Meta) angehängt.

ButtonPress, ButtonRelease Drücken/Loslassen einer Maustaste. Die Tastennummer (1 = linke Taste), die (X/Y)-Koordinate sowie eine Liste von Modifiern werden als Event-Detail übergeben.

MotionNotify Dieses Event signalisiert eine Bewegung der Maus bei gedrückter Maus-Taste. Event-Detail ist das gleiche wie bei ButtonPress.

MenuSelection Dieses Event wird bei Auswahl eines Menü-Eintrags erzeugt. Event-Detail ist dabei der Menü-Pfad, wie er bei der Erzeugung des Eintrags durch `addmenupath()` angegeben wurde.

GetText Dieses Event wird bei Betätigung des OK-Buttons einer Texteingabebox (siehe `gettext()`) erzeugt. Event-Detail ist die eingegebene Zeichenkette. Bei mehrzeiligen Textboxen werden Newlines durch die Zeichenkette \n ersetzt.

GraphNode Dieses Event wird bei Auswahl eines Graph-Knotens in einer Graphbox erzeugt. Event-Detail ist dabei die ID des Graph-Knotens (siehe `addnode()`).

Select Anwort auf einen `select()` bzw. `selectlist()`-Aufruf. Detail ist der NameStack-Inhalt (siehe GL-Referenz).

Coordinate Antwort auf einen `screen2world()`-Aufruf. Detail ist hier die gefragte Weltkoordinate in der Form `X Y`.

Textwidth Antwort auf einen `textwidth()`-Aufruf. Detail ist die Breite des spezifizierten Textes in Weltkoordinaten.

Error eine von `ngltool` generierte Fehlermeldung. Detail ist der Text der Meldung. Beim Abfangen von Fehlermeldungen im Client ist zu beachten, daß Fehlermeldungen verzögert gesendet werden können (z.B. wird die Meldung über ein undefiniertes Bild erst bei Ausführung von `swapbuffers()` ausgegeben), andererseits können manche Kommandos mehrere Fehlermeldungen verursachen (z.B. `flush()`, `swapbuffers()`).

7 GL-Programmierbeispiele

In diesem Kapitel finden sich Lösungen zu typischen Problemstellungen zur Visualisierung von Daten und Prozessen. Die Beispiele befinden sich im Unterverzeichnis `netvisu/NetGLTool/examples` des Quelltext-Verzeichnisbaumes. Üblicherweise würden diese Kommandofolgen aus einem C/C++-Programm heraus an den Server geschickt werden, zum Verständnis ist jedoch die hier gezeigte kompaktere Schreibweise in Form einer Kommandodatei sinnvoller. Die Dateien können z.B. als Eingabe an `netpipe` (siehe Abschnitt 4.1) verwendet werden.

7.1 Ausgabe von Grundfiguren mit verschiedenen Attributen

Die Kommandodatei `figures.gl` demonstriert die grundlegenden Ausgabemöglichkeiten von `ngltool` . Die Argumente für die `defstipple`-Aufrufe wurden hier nicht ausgeschrieben.

```
# attributes.gl
defstipple(Balls,16,16, ...)
defstipple(fence_knitting,16,16, ...)
defstipple(Grey_32_04,16,16, ...)

settitle("figures.gl")

loadimage("echidna.tif","ergb")

prefsize(500,500)
window(0,0,850,850)
viewport(0,0,500,500)

clear()
loadidentity()

pushname("echidna")
putpixels("ergb",10,0,320,200)
popname()
drawstring("echidna.tif",10,200)

pushattributes()
setfontheight(30)
setfontbase("bitstream-charter-bold-r-normal")
drawstring("bitstream-charter-bold-r-normal@30",10,10)
popattributes()

pushattributes()
  color(1.0,1.0,0.0)
  setstipple(fence_knitting)
  rectf(100,100,300,300)
popattributes()


pushattributes()
  color(0.0,0.0,0.0)
  setstipple(Balls)
  rectf(150,150,200,200)
popattributes()

pushattributes()
```

```
  color(1.0,0.1,0.0)
  bgnpolyf()
    vertex(171,342)
    vertex(504,135)
    vertex(792,301)
    vertex(801,585)
    vertex(558,697)
    vertex(513,486)
    vertex(270,490)
  endpolyf()

  color(0.0,0.0,0.0)
  setstipple(None)
  bgnpoly()
    vertex(171,342)
    vertex(504,135)
    vertex(792,301)
    vertex(801,585)
    vertex(558,697)
    vertex(513,486)
    vertex(270,490)
  endpoly()
  color(0.0,0.0,1.0)
  linewidth(5)
  poly((171,342),(504,135),(792,301),(801,585),(558,697),(513,486),(270,490))
popattributes()

setstipple(None)
linewidth(2)
color(0.0,0.0,0.0)
arc(400,600,200,200,360,0)

pushattributes()
  setstipple(Grey_32_04)
  color(0.0,0.7,0.1)
  pushname("Arc1")
  arcf(200,700,350,200,60,240)
  popname()
  setstipple(None)
  linewidth(2)
  color(0.0,0.0,0.0)
  pushname("Arc2")
  arc(200,700,350,200,60,240)
  popname()
popattributes()

swapbuffers()
```

7.2 Aufbau eines Menü-Systems

Menüs lassen sich, ähnlich Verzeichniseinträgen im Dateisystem, beliebig hinzufügen und löschen.

```
addmenupath("File/open")
addmenupath("File/save")
addmenupath("File/quit")
addmenupath("View/new")
addmenupath("Quotes/\"DoubleQuotes\"")
```

```
addmenupath("Quotes/'SingleQuotes'")
addmenupath("Quotes/Backslash:\\")
addmenupath("Options/o1",check)
addmenupath("Options/checkbox")
addmenupath("Options/checkbox/o3",check,0)
addmenupath("Options/checkbox/o4",check,1)
addmenupath("Options/checkbox/o5",check,1)
addmenupath("Options/radiobox")
addmenupath("Options/radiobox/one",radio,1)
addmenupath("Options/radiobox/two",radio)
addmenupath("Options/radiobox/three",radio)
addmenupath("Options/radiobox/four",radio)
addmenupath("View/close")

delmenupath("Options/checkbox/o4")
```

7.3 Ausgabe einer TIFF-Datei in Original-Größe

Um eine 1:1-Abbildung eines Rasterbildes zu erzielen, muß die Window-Viewport-Transformation auf die Dimension des Bildes gesetzt werden. Die Abmessungen des Bildes kann z.B. ermittelt werden, indem man das Bild als `Image`-Objekt instantiiert, wobei der `nodata`-Parameter des Konstruktors als `true` spezifiziert wird (siehe 8.3).

```
prefsize(320,200)
window(0,0,320,200)
viewport(0,0,320,200)
loadimage("echidna.tif","bild1")
putpixels("bild1",0,0,320,200)
swapbuffers()
```

7.4 Aufbau von Makro-Bibliotheken

Durch die Möglichkeit, Display-Listen in ein frei definierbares lokales Koordinatensystem zeichnen zu können, läßt sich leicht eine Makro-Bibliothek aufbauen. Mit diesen Makros kann man nun beliebig komplexe Objekte zusammensetzen. Im unten aufgeführten Quelltext sind identische, wiederholte Kommandosequenzen durch geschweifte Klammern angedeutet, im Eingabestrom für `ngltool` müssen diese ausgeschrieben werden.

```
# transform.gl
prefsize(800,800)
window(-20000,-20000,40000,40000)
viewport(0,0,800,800)

newlist(polygon)
pushattributes()
pushmatrix()
pushname("polygon")
color(1.0,0.1,0.0)
bgnpolyf()
  vertex(-1440,608)
  vertex(1890,-1462)
  vertex(4770,203)
  vertex(4860,3038)
  vertex(2430,4163)
  vertex(1980,2048)
  vertex(-450,2093)
endpolyf()
```

```
color(0.0,0.0,0.0)
bgnpoly()
  vertex(-1440,608)
  vertex(1890,-1462)
  vertex(4770,203)
  vertex(4860,3038)
  vertex(2430,4163)
  vertex(1980,2048)
  vertex(-450,2093)
endpoly()
popname()
popmatrix()
popattributes()
endlist()

newlist("rectangles")
pushattributes()
pushmatrix()
translate(17000,0)
color(0.1,0.8,0.1)
pushmatrix()
translate(1000.0,1000.0)

{ 8 mal wiederholen
rotate(0.0,0.0,0.78539816339744830962)
rectf(0,0,2000,2000)
}

popmatrix()
color(0.8,0.8,0.1)
linewidth(2)
rectf(0,0,2000,2000)
color(0.0,0.0,0.0)
rect(0,0,2000,2000)

popmatrix()
popattributes()
endlist()

newlist("single")
pushmatrix()
translate(5000,0)
calllist("polygon")
popmatrix()
endlist()

newlist("all")
pushmatrix()

{ 9 mal wiederholen

calllist("single")
calllist("rectangles")

rotate(0.0,0.0,0.31415)
calllist("single")
```

```
}

popmatrix()
endlist()

clear()
loadidentity()
translate(-10000,-10000)
rotate(0.1)
pushattributes()
linewidth(5)
color(1.0,0.648,0.0)
rect(-8000,-8000,16000,16000)
popattributes()
scale(0.4,0.4)
calllist("all")
swapbuffers()
```

7.5 Zeichnen von maussensitiven Objekten

Folgende Kommandofolge verknüpft den Bezeichner "inside" mit einem Polygon. Wenn der Befehl `select()` mit einer Koordinate innerhalb des Polygons aufgerufen wird, wird dieser Bezeichner als Detail eines Select-Events zurückgeliefert.

```
prefsize(300,300)
backbuffer()
clear()
window(0,0,8500,8500)
viewport(0,0,300,300)
color(1.0,0.1,0.0)
pushname(inside)
bgnpolyf()
  vertex(1710,3420)
  vertex(5040,1350)
  vertex(7920,3015)
  vertex(8010,5850)
  vertex(5580,6975)
  vertex(5130,4860)
  vertex(2700,4905)
endpolyf()
color(0.0,0.0,0.0)
bgnpoly()
  vertex(1710,3420)
  vertex(5040,1350)
  vertex(7920,3015)
  vertex(8010,5850)
  vertex(5580,6975)
  vertex(5130,4860)
  vertex(2700,4905)
endpoly()
popname()
swapbuffers()
```

7.6 Ausgabe von maussensitivem Text mit verschiedenen Attributen (à la Hypertext)

```
prefsize(400,200)
window(0,0,400,200)
viewport(0,0,400,200)
clear()
setfontheight(30)
setfontbase("bitstream-charter-bold-r-normal")
drawstring("Dieser Text ist ",10,40)
pushattributes()
pushname("sensitiv")
color(0.0,0.0,1.0)
drawstring("Maus-Sensitiv",10,72)
popname()
popattributes()
drawstring("Der hier nicht",10,104)
swapbuffers()
```

7.7 Zeichnen eines Rubber-Rectangles auf einem statischem Hintergrund

Zuerst wird in den Hintergrundpuffer etwas ausgegeben:

```
prefsize(300,300)
backbuffer()
clear()
window(0,0,8500,8500)
viewport(0,0,300,300)
color(1.0,0.1,0.0)
pushname(inside)
bgnpolyf()
  vertex(1710,3420)
  vertex(5040,1350)
  vertex(7920,3015)
  vertex(8010,5850)
  vertex(5580,6975)
  vertex(5130,4860)
  vertex(2700,4905)
endpolyf()
color(0.0,0.0,0.0)
bgnpoly()
  vertex(1710,3420)
  vertex(5040,1350)
  vertex(7920,3015)
  vertex(8010,5850)
  vertex(5580,6975)
  vertex(5130,4860)
  vertex(2700,4905)
endpoly()
popname()
swapbuffers()
```

In einer Schleife kann nun ein variierbares Rechteck (Rubber-Rectangle) wie folgt gezeichnet werden:

```
swapbuffers()
```

```
frontbuffer()
clearlist()
rect(startx,starty,endx,endy)
flush()
```

Die Variablen *startx,starty,endx,endy* werden durch das Anwender-Programm ermittelt (durch Auswertung von Maus-Events). Dadurch muß der Hintergrund nur einmal gezeichnet werden: `swapbuffers()` macht diesen in jedem Schleifendurchgang sichtbar, der `rect()`-Befehl zeichnet direkt ins Fenster. Der Aufruf von `clearlist()` verhindert, daß `ngltool` alle Rechtecke in der Refresh-Liste des Front-Buffers aufsammelt. Der Aufruf von `flush()` erzwingt schließlich eine unmittelbare Ausgabe des Rechtecks in den Frontbuffer.

8 Programmierreferenz

Die nachfolgenden Abschnitte beschreiben den Aufbau von NetGLTool und dokumentieren die C++-API der zugehörigen Libraries. Um mit

8.1 Struktur des Programmpakets

Das ganze Programmpaket besteht, neben dem Hauptprogramm `ngltool`, aus den Libraries, auf denen `ngltool` aufbaut. Eine grundlegende Komponente, die LEDA-Library des MPI Saarbrücken, wird in deren Manual ausführlich beschrieben. Zusätzlich zu den nachfolgend beschriebenen Libraries werden folgende Pakete benutzt:

- Die JPEG-Library in der Version 5b
- Die TIFF-Library von Sam Leffler, Version 3.3 BETA
- Die Iris RGB library "libimage" von Paul Haeberli
- Das Motif UI Toolkit, Versionen 1.2 bzw. 2.0

8.1.1 MotifApp (Motif-Einbindung)

MotifApp ist eine Sammlung von GUI-Klassen, die zusammen ein Framework zur Generierung von Motif-Oberflächen bilden. Die Bibliothek baut auf den Konzepten und dem Source-Code des von Douglas A. Young entwickelten Frameworks auf, das in seinem Buch [YNG] ausführlich beschrieben wird. Es wurden viele neue Klassen hinzugefügt und große Teile des übernommenen Codes modfiziert. MotifApp stellt die Basisklassen für alle hier beschriebenen Interface-Objekte bereit.

8.1.2 EasyDraw

EasyDraw ist eine Sammlung von C++-Klassen, die zusammen in ein GUI- Interface-Objekt einfließen. Dieses Interface-Objekt (Klasse `DrawingArea`) beinhaltet C++-Methoden zur Ausgabe von Graphiken auf einer Zeichenfläche. Die meisten Befehle der GL-Kommandosprache haben eine Entsprechung als Methode in dieser Klasse. Weiter beinhaltet EasyDraw eine Applikations-Klasse, die als Basis für komplette GUIs verwendet werden kann (Klasse `GraphicsTool`). Sowohl `DrawingArea` als auch `GraphicsTool` bauen auf dem MotifApp-Framework auf (d.h. benutzen die dort definierten Basisklassen).

8.1.3 Image

Die Klasse Image dient zur Handhabung von Rasterbildern innerhalb des EasyDraw-Frameworks. Image beinhaltet Methoden zum Laden, Abspeichern, Skalieren sowie pixelweisem Zugriff von TIFF, JPEG und IRIS-RGB Bildern. In der EasyDraw-library findet sich eine Interface-Klasse (Klasse `PictureManager`), die ein Image so aufbereitet, daß es unter X-Windows auf dem Bildschirm dargestellt werden kann.

8.1.4 libsocket++

Die Library libsocket++ ist die Netzwerkkomponente von NetGLTool. Sie beinhaltet Klassen zum einfachen Aufbau von Netzwerkverbindungen. Da die Klassen für die Einrichtung einer Netzwerkverbindung zu einem `ngltool` -Server und auch allgemein zur Netzwerkprogrammierung eingesetzt werden kann, wird

8.1.5 libGL

Die Library libGL beinhaltet den Parser für die Kommandosprache von NetGLTool.

8.2 EasyDraw

Die EasyDraw-Library umfaßt eine Reihe von Klassen, die sich alle um ein Zeichenflächen-Objekt (`DrawingArea`) drehen. Die Klasse `DrawingArea`, die eine GUI-Komponente innerhalb des MotifApp-Frameworks darstellt, enthält Methoden, die zum großen Teil direkte Pendants zu Befehlen aus der Graphik-Kommandosprache darstellen. Eine weitere Toplevel-Klasse stellt `GraphicsTool` dar, sie faßt eine DrawingArea mit einem Menü-Balken und einem Scale-Widget in einem lauffähigen Anwendungsprogramm zusammen. Der Benutzer bildet Unterklassen dieser Klasse mit spezialisierten Ein-/Ausgabe-Methoden und instantiiert ein Objekt davon. Der Netzwerk-Server `ngltool` ist nach diesem Konzept aufgebaut.

8.2.1 DrawingArea

Die Klasse DrawingArea stellt die Funktionalität bereit, die von `ngltool` zur Realisierung der Graphik-Kommandos benötigt wird. Jedes GL-Graphik-Kommando hat seine Entprechung in einer Methode dieser Klasse. Weiter existieren hier Methoden, die die Lupen-Funktionalität sowie die Event-Handler für Neuzeichnen, Größenänderung und Eingabe realisieren.

8.2.2 GraphicsTool

Ein Objekt der Klasse `GraphicsTool` realisiert ein komplettes User-Interface mit Menübalken, Zeichenfläche und einem Scale-Widget, mit dem man einen Zoom-Faktor für die Zeichenfläche einstellen kann. Ein `GraphicsTool` enthält ein oder mehrere `GraphicsPanel`, dies ist im wesentlichen ein Container für die User-Interface-Objekte. Die Klasse `GraphicsPanel` deklariert zwei virtuelle Methoden, `redraw()` und `input()`, die jeweils aufgerufen werden, wenn der Inhalt der Zeichenfläche neu aufgebaut werden muss oder der Benutzer mit der Maus oder Tastatur eine Eingabe auf der Zeichenfläche erzeugt. Mit der Methode `canvas()` kann auf die eigentliche Zeichenfläche zugegriffen werden.

8.2.3 PictureManager

Diese Klasse dient zur Konvertierung der (nicht X-Windows-spezifischen) Image-Objekte in Datenstrukturen, die unter X-Windows auf dem Bildschirm dargestellt werden können. Zu diesem Zweck wird auf den Bilddaten eine Farbraumkonvertierung mit Dithering durchgeführt; die resultierende Pixmap kann mit dem XCopyArea (siehe [ORA2])-Aufruf direkt in ein X-Drawable (Fenster oder andere Pixmap) kopiert werden. Mittels einer Methode in der Image-Klasse kann auf diese Pixmap zugegriffen werden. Als Zwischenstufe wird eine XImage-Datenstruktur erzeugt, auch auf diese kann mittels einer Methode zurückgegriffen werden.

8.3 Image

Die Image-library ist zusammen mit EasyDraw entwickelt worden, um die Handhabung von Raster-Bildern zu ermöglichen. Sie kann (und wird) jedoch auch ohne die anderen EasyDraw-Komponenten verwendet werden.
Grundlage ist die Aufspaltung des Raster-Bildes in einen oder mehrere *Kanäle*, die jeweils einen rechteckigen Bereich (2-Dimensionales Array) von 1-Byte Werten darstellen. Die Bedeutung der Kanäle ist Applikations-abhängig, aber üblich bzw. denkbar sind z.B.:

- 1 Kanal für Grau-Werte oder Schwarz-Weiß.
- 3 Kanäle für Rot, Grün, Blau (24 Bit Truecolor-Bild)
- 6 Kanäle für RR BB GG (48 Bit Truecolor-Bild, 16 Bit/Kanal)
- 4 Kanäle für Rot, Grün, Blau, Alpha-Wert (Gewichtung).
- beliebige Anzahl von Kanälen z.B. für Volumendaten.

Die Image-Klasse bietet Methoden zum Laden und Speichern dieser Raster-Bilder in verschiedenen Formaten (momentan: TIFF, JPEG, Iris RGB). Eine weitere Klasse, `ImageRow`, ermöglicht einen zeilenorientierten Zugriff auf die Bilddaten. Außerdem exisitiert eine Reihe von template-Funktionen, die einen kopierenden Zugriff auf einzelne Rasterzeilen des Bildes bieten.

Ein besonderes Feature ist die Verwendung eines Cache für die Verwaltung der Bilddaten. Ein Bilddaten-Block ist in diesem Zusammenhang ein Objekt der Klasse `ImageData`. Es beinhaltet die eigentlichen Pixel-Werte, einen Zeiger auf den 'Besitzer' (d.h. das Image-Objekt zu dem es gehört), einen Zeiger auf benutzungsspezifische Daten, die von einer Image-Unterklasse verwaltet werden können (z.B. X-Pixmap im Fall von DisplayImage) sowie die minimalen Geometrie-Informationen (Breite, Höhe, Tiefe/Anzahl der Kanäle). Der Cache wird durch ein einzelnes Objekt der Klasse `ImageCache` repräsentiert. Dieses Objekt wird bei Instantiierung der globalen Variable `theGlobalImageCache` zugewiesen. Der Cache hat die beiden Grenzen

- Maximale Anzahl der Bilddaten-Blöcke, die im Speicher gehalten werden können (Compile-time-option)
- Maximale Speichermenge, die von allen Bilddaten-Blöcken zusammen belegt werden darf (Konstruktor-Argument von `ImageCache`).

Wird eine dieser Grenzen überschritten, so werden Blöcke nach einem LRU-Schema in eine temporäre Datei verdrängt. Bei einem späterem Zugriff werden die verdrängten Blöcke wieder in den Speicher geladen.
Instanzen der Klasse `Image` können nun solche Bilddaten-Blöcke referenzieren. Ein Objekt dieser Klasse enthält neben einem Zeiger auf die Original-Bilddaten einen Zeiger auf ein skaliertes Bild. Die Methode `scale(int width,int height)` erzeugt dieses skalierte Bild. Falls die Skalierung bereits zu einem früherem Zeitpunkt erzeugt wurde, wird im Cache darauf zugegriffen, wodurch sich die Ausführung beträchtlich beschleunigt. (Abbildung 4).

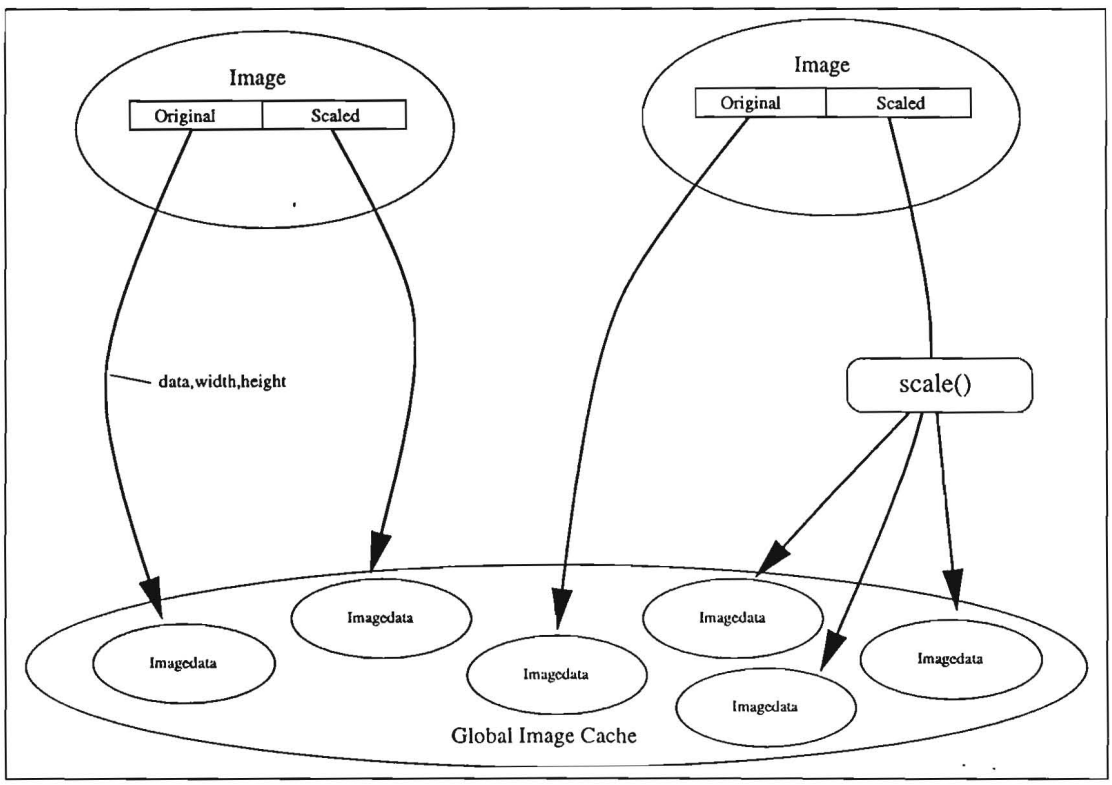

Abbildung 4: Der Bilddaten-Cache

Die `ImageRow`-Zugriffsfunktionen, die Farbwerte an einer Pixel-Koordinate liefern, beziehen sich normalerweise auf das skalierte Bild. Diese Vorgehensweise rührt von der ursprünglichen Anwendung dieser Klasse, der Visualisierung der Bilddaten, her. Es gibt jedoch auch die Möglichkeit, auf die unskalierten Rohbilddaten zuzugreifen (mittels eines Konstruktor-Arguments für `ImageRow`).
Ferner exisitiert noch ein Zeiger auf ein "ImageExtensionRecord", dies ist ein Container für Bilddateityp-spezifische Daten. Bisher ist dies nur für das TIFF-Format realisiert.

8.3.1 Methoden der Klasse ImageData

Zur Beachtung: diese Klasse stellt keinen Konstruktor bereit, da die Objekte ausschließlich vom Cache erzeugt werden dürfen.
Im folgenden wird der Typ `Dimension` als Abkürzung für unsigned short benutzt.

Dimension width() Breite des Blocks in Pixel

Dimension height() Hoehe des Blocks in Pixel

Dimension nChannels() Anzahl der Kanäle

Image *owner() Image-Objekt, zu dem dieser Block gehört

Boolean cached() Flag: wird dieser Block noch vom Cache verwaltet (siehe unten)

ImageDataExtension *extension() Zugriff auf das Extension-Record des Bilddaten-Blocks.

Die Klasse ImageDataExtension ist eine abstrakte Basisklasse, die die beiden virtuellen Methoden `swapout()` und `restore()` bereitstellt. Die Methode `swapout()` wird beim Auslagern des Bilddaten-Blocks aufgerufen, `restore()` beim Wieder-Einlagern. Mit diesem Mechanismus kann der Benutzer bei den beiden Vorgängen zusätzliche Aktionen ausführen lassen (z.B. Freigeben/Erzeugen der Pixmap für die Visualisierung).
Da die Klasse ImageCache ihre Aufgabe ohne weitere Intervention des Programmierers erledigt, sei hier nur erwähnt, daß sich mit der Methode `uncache(ImageData *)` ein Block aus der Verwaltung herauslösen läßt. Der Block wird nicht mehr zum aktuellen Speicherbedarf des Cache hinzugezählt und muß vom Benutzer explizit (mit `delete`) freigegeben werden. Im Fall, daß ein Bilddaten-Block bei der Alloziierung größer ist als die Cache-Grenze, wird dieser Block direkt aus der Cache-Verwaltung herausgenommen (das Löschen von skalierten Bilddaten erledigt in diesem Fall die Methode `scale()`).

8.3.2 Methoden der Klasse Image

Hier folgt eine Erläuterung der Image-Methoden, die für die Benutzung durch den Anwender vorgesehen sind.

Image() (Konstruktor) erzeuge ein leeres Bild. Es werden keine Bilddaten alloziiert oder Geometrie-Informationen gesetzt.

Image(Dimension w,Dimension h,Dimension z) (Konstruktor) Es werden Bilddaten mit der gegebenen Geometrie alloziiert. Der Parameter z gibt die Anzahl der Kanäle an.

Image(const char *file) (Konstruktor) erzeuge Bild und lade Bilddaten aus der gegebenen Bilddatei.

~Image() Destruktor. Alloziierter Speicher wird freigegeben.

const char * filename() const Gebe den Dateinamen der Bilddatei zurück.

void filename(const char *newFilename) Setze den Dateinamen der Bilddatei.

const Boolean isValid() const Flag: wurde eine Bilddatei korrekt eingelesen?

int imageDataType() const Interpretation der Bilddaten. Der Rückgabewert ist einer der Werte `IMAGEDATA_NONE`, `IMAGEDATA_RGB`, `IMAGEDATA_CMYK`, `IMAGEDATA_GRAYSCALE`, `IMAGEDATA_CMAP`, `IMAGEDATA_PACKED` aus dem Aufzählungstyp `Image::ImageDataType`. Für diesen Slot gibt es keine schreibende Zugriffs-Methode, da er konsistent mit dem Format der existierenden Bilddaten sein muß.

int imageFileType() const Typ der Bilddatei, aus der das Bild gelesen wurde. Der Rückgabewert ist einer der Werte `IMAGEFILE_NONE`, `IMAGEFILE_JPEG`, `IMAGEFILE_IRIS`, `IMAGEFILE_TIFF` aus dem Aufzählungstyp `Image::ImageFileType`.

void imageFileType(ImageFileType newType) Setze den Typ der Bilddatei, der dann beim nächsten Abspeichern beachtet wird. Der Parameter ist einer der Werte aus Image::ImageFileType.

int read(const char *,Boolean nodata=0) lade Bilddaten aus der gegebenen Bilddatei. Das Flag `nodata` gibt an, ob nur der Header der Bilddatei oder auch die Bilddaten gelesen werden sollen. Dies kann benutzt werden, um die Geometrie-Informationen der Bilddatei zu lesen. Die Methode bestimmt selbständig das Format der Bilddatei.

int write(const char *) Bilddaten in Bilddatei speichern. Das Format (TIFF, JPEG oder Iris RGB) ergibt sich aus imageFileType().

ImageData *original() Direktzugriff auf den Bilddatenblock, der den Originaldaten zugeordnet ist.

ImageData *scaled() Direktzugriff auf den Bilddatenblock, der der eingestellten Skalierung zugeordnet ist.

const Dimension dataWidth() const gebe die Breite der alloziierten / gelesenen Bilddaten zurück.

const Dimension dataHeight() const gebe die Höhe der alloziierten / gelesenen Bilddaten zurück.

unsigned bitsPerPixel() const Anzahl der Bits pro Pixelwert.

unsigned nChannels() const Anzahl der allozierten Kanäle.

unsigned bytesPerRow() const Anzahl der Bytes, die zur Speicherung einer Rasterzeile benötigt werden.

const Dimension scaledWidth() const Breite des skalierten Bildes.

const Dimension scaledHeight() const Höhe des skalierten Bildes.

const ImageExtensionRecord *extensionRecord() const Zugriffsmethode für die ImageExtension (momentan nur bei TIFF-Bildern benutzt).

double aspect() const Liefert den Aspect Ratio des Bildes, d.h. das Verhältnis Breite zu Höhe des skalierten Bildes.

void scale(Dimension newX,Dimension newY,Boolean cacheable) Erzeuge Bilddaten für ein skaliertes Bild. Das neue Bild hat die Abmessungen (newX,newY). Bei einer Vergrösserung des Bildes werden die Bilddaten mit einem blur-Filter nachbearbeitet. Der Algorithmus stammt von Raul Rivero [RIV]. Der Parameter `cacheable` (default-Wert True) spezifiziert, ob die Bilddaten im Cache verwaltet werden sollen. Mit dieser Methode kann auch der Aspekt Ratio des Bildes verändert werden.

byte Pixel(Position x,Position y,unsigned z) liefert den Pixel-Wert an der Stelle (x, y) , Kanal z.

void Pixel(Position x,Position y,unsigned z,byte newValue) setzt den Pixel-Wert an der Stelle (x, y) , Kanal z auf $newValue$.

byte Pixel(Position x,Position y,unsigned z liefert den Pixel-Wert an der Stelle (x, y) , Kanal z.

void Pixel(Position x,Position y,unsigned z,byte newValue setzt den Pixel-Wert an der Stelle (x, y) , Kanal z auf $newValue$.

void allocColorMap() Es wird eine Farbtabelle mit $2^{bitsPerPixel}$ Einträgen angelegt. Diese Farbtabelle wird nur beim Lesen und Schreiben von Bildern des Typs `IMAGEDATA_CMAP` beachtet.

unsigned short colorMapSize() const Anzahl der Farbtabellen-Einträge zurückliefern.

RGBPixel colorMapEntry(unsigned i) liefere i-ten Farbtabelleneintrag als RGB-Wert zurück. `RGBPixel` ist eine Klasse, die in in der Header-Datei zu libImage definiert wird und im wesentlichen einen Container für einen RGB-Wert dárstellt.

void colorMapEntry(unsigned i,RGBPixel p) Setze i-ten Farbtabelleneintrag auf einen neuen RGB-Wert.

8.3.3 Methoden der Klasse ImageRow

Objekte der Klasse ImageRow stellen "intelligente Zeiger" auf eine Zeile innerhalb eines Image-Objektes dar. Man kann über den array-operator ("`[ ]`") auf die einzelnen Kanäle eines Pixels zugreifen, den Zeiger mit den increment,decrement-Operatoren in der Zeile vorwärts/rückwärts bewegen sowie auf beliebige Zeilen innerhalb des Bildes positionieren. Bei Zugriffen auf nichtexistente Kanäle (z.B. Kanal 2 bei einem Graustufenbild) wird der Wert in Kanal 0 zurückgegeben, dadurch wird eine einheitliche Behandlung aller Bildtypen erleichtert.

ImageRow(Image *image,int y=0,Boolean fromSource=False) Konstruktor, Argumente sind das Image-Objekt, die Zeilennummer, sowie ein Flag, ob das skalierte oder unskalierte Bild benutzt werden sollen.

byte &operator [](unsigned short n) Zugriff auf n-ten Bildkanal. Rückgabewert ist der Pixelwert des n-ten Kanals an der aktuellen Position in der Zeile.

operator++() auf nächsten Pixel innerhalb der Zeile positionieren.

operator--() auf vorhergehenden Pixel innerhalb der Zeile positionieren.

operator()(int newY) auf Anfang der Zeile newY positionieren.

8.4 libsocket++

Die library libsocket++ stellt einen C++-Layer zur BSD Socket-API in Form von 3 Klassen dar. Zweck des Klasseninterfaces ist es, eine einfache Möglichkeit zum Aufbau von Netzwerk-Servern und -Clients zu bieten. Der Benutzer der Klassenbibliothek wird hier auf ein konkretes Konzept festgelegt, nämlich

- zuverlässige, nicht-paketorientierte Verbindungen (TCP-Streams)
- synchrone Multiplex-Server: Der Server tätigt die I/O von mehreren Clients quasi gleichzeitig, und zwar ohne Unterteilung in mehrere Unix-Prozesse. Die Datenübertragung geschieht synchronisiert, d.h. es wird erst dann etwas übertragen, wenn die jeweilige Gegenseite Bereitschaft dazu zeigt.

Auf Unix-Ebene sieht das so aus, daß ein Server-Programm gestartet wird und dann auf ankommende Verbindungen wartet. Diese Verbindungen werden von Client-Programmen initiiert; der Server-Prozeß nimmt diese Verbindungen an. Beide Seiten können dann über ein C++-iostream-Objekt Daten austauschen. Der Server läuft dabei üblicherweise in einer Endlosschleife und prüft, welche der angemeldeten Clients zur Ein-/Ausgabe bereit sind. Mit diesen Clients können dann (verzögerungsfrei, d.h. ohne Blockade des Server-Prozesses) Daten ausgetauscht werden. Beim Verbindungsaufbau spielt die TCP-Port-Nummer eine Rolle, dies ist eine 16-Bit-integer-Zahl, die zur Adressierung eines Netzwerk-Dienstes auf einem Rechner benutzt wird. Der Server-Prozeß wartet unter dieser Port-Nummer auf ankommende Verbindungen. Die Clients benutzen, neben der IP-Adresse des Ziel-Rechners, diese Zahl, um eine Verbindung zum gewünschten Netzwerk-Server zu initiieren.
Die beteiligten Klassen sind im einzelnen:

Server Ein Objekt dieser Klasse verwaltet alle ankommenden Verbindungen. Methoden der Klasse prüfen, ob neue Client-Verbindungen angefordert werden und welche dieser Verbindungen zur Ein-/Ausgabe bereit sind.

serverstream Stream-Verbindung vom Server-Prozeß zu einem Client-Prozeß. Objekte dieser Klasse werden ausschließlich von einem `Server`-Objekt instantiiert. Wenn im Kontext der Klasse Server von einem "Client" die Rede ist, so ist hiermit ein Objekt der Klasse `serverstream` gemeint.

clientstream Stream-Verbindung von einem Client-Prozeß zum Server-Prozeß.

8.4.1 Server

Server(unsigned short port) Konstruktor der Klasse Server; erzeuge Server-Objekt mit der angegebenen Port-Nummer.

int checkNewClients(long timeout = 0) Warte maximal *timeout* Milli-Sekunden auf neue Verbindungen. Für jede eingehende Verbindung wird ein `serverstream`-Objekt erzeugt, das mit `getNextClient()` abgeholt werden kann.

serverstream *getNextClient() hole nächste, von einem Client-Prozeß angeforderte Verbindung. Die Instantiierung des `serverstream` erfolgt in `checkNewClients()`. Diese Methode registriert den `serverstream` auch gleichzeitig für die Überprüfung durch die Methode clientsReady().

void clientsReady(list<serverstream *> &readable,
list<serverstream *> &writable,
long timeout = 0) Mit dieser Methode kann zu einem beliebigem Zeitpunkt geprüft werden, welche der Clients zum Lesen und / oder Schreiben bereit sind. Es wird maximal *timeout* Milli-Sekunden gewartet. Diese Methode ist zum verzögerungsfreien Lesen und Schreiben von / zu mehreren Client-Prozessen notwendig. Nur die mit getNextClient() registrierten `serverstream`-Objekte werden von dieser Methode geprüft.

void disconnect(serverstream *) Verbindung zum Client abbrechen.

int waiting() liefert 1 gdw. der Server erfolgreich initialisiert werden konnte und bislang kein Fehler aufgetreten ist.

int socket() liefert den UNIX-Dateideskriptor, der mit der Server-Addresse assoziiert ist.

8.4.2 socketbuf

Die Klasse `socketbuf` ist eine Unterklasse von streambuf und hat in diesem Zusammenhang für die Klassen `clientstream` und `serverstream` die gleiche Aufgabe wie `filebuf` für `fstream`: sie ist dafür zuständig, Zeichen von einer Eingabequelle zu holen bzw. in eine Senke zu schreiben. Quelle und Senke sind hier die Netzwerkverbindung, realisiert durch einen Socket. Die Klasse implementiert im wesentlichen neue Versionen der virtuellen Methoden `int underflow()` und `int overflow(int)`, die hier für die synchronisierte Ein-/Ausgabe über die Netzwerkverbindung zuständig sind. Die Synchronisierung erfolgt über die Methoden `int gavailable(long)` und `int pavailable(long)`, die feststellen können, ob der benutzte Netzwerk-Socket innerhalb einer gegebenen Zeitspanne für die Datenübertragung bereit ist.

8.4.3 clientstream

Ein Objekt der Klasse `clientstream` repräsentiert die Client-Seite der Netzwerkverbindung. Das Klassenprotokoll gleicht dem von `fstream`, außer dem Konstruktor:

clientstream(const char *hostname, int portno) Instantiiere eine Netzwerkverbindung zum Rechner `hostname` und zum Service mit der Port-Nummer `portno`. Der resultierende Stream ist Instanz einer Unterklasse von `iostream`, stellt also eine Zwei-Wege-Verbindung dar.

8.4.4 serverstream

Ein Objekt der Klasse `serverstream` repräsentiert die Server-Seite der Netzwerkverbindung. Das Klassenprotokoll gleicht dem von `fstream`, es existiert jedoch kein `public`-Konstruktor (siehe Beschreibung zur Klasse `Server`).

9 Programmierbeispiele

9.1 Beispiele zu libsocket++

Die beiden Beispielprogramme realisieren eine einfache Einrichtung zur Übertragung von Dateien über eine Netzwerkverbindung. Der Server `file_send` wartet auf ankommende Verbindungen. Das Client-Programm initiiert eine Verbindung zum Server und sendet einen Dateinamen in Form einer ASCII-Zeile. Der Server öffnet daraufhin diese Datei auf dem remote-System und überträgt deren Inhalt an das Client-Programm. Das Client-Programm kopiert die ankommenden Daten in eine Datei auf dem lokalen System.

9.2 Der Server file_send

Der Server besteht im wesentlichen aus der Prozedur `ServerLoop`, die in einer Endlosschleife Verbindungen akzeptiert und abarbeitet. Während eines Schleifendurchlaufes wird zuerst eine Liste von neuen Verbindungen angelegt (dies behandelt den Fall, daß mehrere Clients gleichzeitig einen Verbindungsaufbau versuchen) und dann für jede Verbindung zuerst ein Dateiname

gelesen, die Datei geöffnet und blockweise an den Client abgeschickt. Hier wird nur der einfache Fall behandelt, daß die Daten an einem Stück abgesendet werden können. Sobald einer der Clients die Übertragung blockiert, so blockiert auch der Server (in `serverstream::write()`).

```
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <LEDA/list.h>
#include "socketstream.h"

#include <fstream.h> // Fuer ifstream
#include <string.h>

#define PORT_NUMBER 8999
#define TIMEOUT 1000000L
#define BUFFER_SIZE 1024

//
// demo server:
//


void ServerLoop( Server *server )
{

  if (! server->waiting() )
    exit(1);

  char buffer[BUFFER_SIZE];
  list<serverstream *> clients;
  while( server->waiting() )
    {
      // Neue Verbindungen annehmen
      for (int i = server->checkNewClients(TIMEOUT); i ; i--)
        clients.append(server->getNextClient());

      serverstream *client;
      while ( (clients.length()) )

        // forall( client, clients )
        {
          char filename[PATH_MAX];
          client = clients.pop();
          if( client->good() )
            {
              // Dateinamen lesen
              client->getline( filename, PATH_MAX, '\n' );
              cout << '[' << client->rdbuf()->fd() << "] "
                   << filename << endl;
            }
          else
            {
              client->close();
              continue;
            }

          //  Datei an den Client schicken
          if( strlen (filename) && !access(filename,R_OK))
            {
              ifstream from( filename );
              // Datei einlesen
```

```
            while (from.good())
              {
                from.read(buffer,BUFFER_SIZE);
                client->write(buffer,from.gcount());
              }

            client->close();
          }
        else
          cerr << "invalid file name " << buffer << endl;
      }
  } // end of while()
}

main (int argc, char *argv[])
{
  int port;
  if (argc > 1)
    port = atoi(argv[1]);
  else
    port = PORT_NUMBER;


  ServerLoop( new Server(port) );
}
```

9.3 Client file_recv

Das Client-Programm besteht im wesentlichen aus der Prozedur ReceiveFile, die den Namen der Remote-Datei an den Server absendet und danach die ankommenden Daten blockweise von der Netzwerkverbindung in eine lokale Datei kopiert.

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "socketstream.h"

#include <fstream.h> // Fuer ofstream

#define BUFFER_SIZE 1024

void
ReceiveFile(const char *remoteFileName,
            const char *localFileName,
            clientstream *connection)
{
  if (!connection->good())
    {
      perror("clientstream");
      exit(1);
    }

  // lokale Datei oeffnen
  ofstream File( localFileName );
  if (!File.good())
    {
      perror("ofstream");
      exit(1);
    }
```

```
  char buffer[BUFFER_SIZE];

  // Dateinamen absenden
  *connection << remoteFileName << endl;
  while (*connection)           // bis EOF
    {
      // einen Block vom Server einlesen
      connection->read(buffer,BUFFER_SIZE);
      // abspeichern
      File.write(buffer,connection->gcount());
    }
}

main (int argc, char *argv[])
{
  if (argc != 5) {
    cerr << "usage:" << argv[0]
         << " remote-file local-file hostname port" << endl;
    exit(1);
  }
  ReceiveFile(argv[1],argv[2],new clientstream( argv[3],atoi(argv[4]) ));
}
```

9.4 Ein Beispiel-Client

Das im folgenden vorgestellte C++-Programm realisiert eine sehr simple Version des im OMEGA-Projektumfeld eingesetzten Dokumenten-Viewers `gucky`. Das Programm visualisiert die Bitmap eines eingescannten Dokuments sowie Text-Segmentierungs- und -Erkennungsergebnisse, die vor der Visualisierung in einer sog. Layout-Datei abgelegt wurden.
Das Programm öffnet eine Netzwerkverbindung zu einem `ngltool`-Server, lädt ein OMEGA-Layout und schickt daraufhin eine Serie von Graphikkommandos an den Server, die das Layout im Graphikfenster darstellen. Das Programm erwartet als Kommandozeilen-Argumente den Rechner-Namen und die Port-Nummer des `ngltool`-Servers sowie den (absoluten) Pfadnamen des OMEGA-Blackboards.
Der Quelltext ist auf im Quelltextverzeichnis unter `netvisu/example` zu finden.

Literatur

[ORA1] Adrian Nye, X Windows Programming Guide, O'Reilly Associates 1992

[ORA2] Adrian Nye, X Windows Reference Guide, O'Reilly Associates 1992

[ORA3] Adrian Nye, X Windows Users Guide, O'Reilly Associates 1992

[ORA4] Adrian Nye, Tim O'Reilly, X Toolkit Intrinsics Programming Manual, O'Reilly Associates 1992

[ORA6a] Dan Heller, Motif Programming Guide, O'Reilly Associates 1994

[ORA6b] Dan Heller, Paula M. Ferguson, Motif Reference Guide, O'Reilly Associates 1994

[YNG] Douglas A. Young, Programming with Motif and C++, Prentice Hall 1992

[RIV] Paul Rivero, LUG User's Manual, Universidad de Oviedo, 1994

[FHFD] Feiner, Hughes, Foley, Van Dam, Computer Graphics: Principles & Practice, 1993

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210
e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-
sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1997

RR-97-01
Erica Melis, Claus Sengler
Analogy in Verification of State-Based Specifications: First Results
12 pages

1996

RR-96-06
Claus Sengler
Case Studies of Non-Freely Generated Data Types
200 pages

RR-96-05
Stephan Busemann
Best-First Surface Realization
11 pages

RR-96-04
Christoph G. Jung, Klaus Fischer, Alastair Burt
Multi-Agent Planning
Using an *Abductive* EVENT CALCULUS
114 pages

RR-96-03
Günter Neumann
Interleaving
Natural Language Parsing and Generation Through Uniform Processing
51 pages

RR-96-02
E.André, J. Müller , T.Rist:
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

RR-96-01
Claus Sengler
Induction on Non-Freely Generated Data Types
188 pages

1995

RR-95-20
Hans-Ulrich Krieger
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

RR-95-19
Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger
Distributed Parsing With HPSG Grammar
20 pages

RR-95-18
Hans-Ulrich Krieger, Ulrich Schäfer
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210
e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-
sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1997

RR-97-01
Erica Melis, Claus Sengler
Analogy in Verification of State-Based Specifications: First Results
12 pages

1996

RR-96-06
Claus Sengler
Case Studies of Non-Freely Generated Data Types
200 pages

RR-96-05
Stephan Busemann
Best-First Surface Realization
11 pages

RR-96-04
Christoph G. Jung, Klaus Fischer, Alastair Burt
Multi-Agent Planning
Using an *Abductive*
EVENT CALCULUS
114 pages

RR-96-03
Günter Neumann
Interleaving
Natural Language Parsing and Generation
Through Uniform Processing
51 pages

RR-96-02
E.André, J. Müller , T.Rist:
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

RR-96-01
Claus Sengler
Induction on Non-Freely Generated Data Types
188 pages

1995

RR-95-20
Hans-Ulrich Krieger
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

RR-95-19
Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger
Distributed Parsing With HPSG Grammar
20 pages

RR-95-18
Hans-Ulrich Krieger, Ulrich Schäfer
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

RR-95-17
Hans-Ulrich Krieger
Classification and Representation of Types in TDL
17 pages

RR-95-16
Martin Müller, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-15
Joachim Niehren, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-14
Joachim Niehren
Functional Computation as Concurrent Computation
50 pages

RR-95-13
Werner Stephan, Susanne Biundo
Deduction-based Refinement Planning
14 pages

RR-95-12
Walter Hower, Winfried H. Graf
Research in Constraint-Based Layout, Visualization, CAD, and Related Topics: A Bibliographical Survey
33 pages

RR-95-11
Anne Kilger, Wolgang Finkler
Incremental Generation for Real-Time Applications
47 pages

RR-95-10
Gert Smolka
The Oz Programming Model
23 pages

RR-95-09
M. Buchheit, F. M. Donini, W. Nutt, A. Schaerf
A Refined Architecture for Terminological Systems:
Terminology = Schema + Views
71 pages

RR-95-08
Michael Mehl, Ralf Scheidhauer, Christian Schulte
An Abstract Machine for Oz
23 pages

RR-95-07
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt
The Complexity of Concept Languages
57 pages

RR-95-06
Bernd Kiefer, Thomas Fettig
FEGRAMED
An interactive Graphics Editor for Feature Structures
37 pages

RR-95-05
Rolf Backofen, James Rogers, K. Vijay-Shanker
A First-Order Axiomatization of the Theory of Finite Trees
35 pages

RR-95-04
M. Buchheit, H.-J. Bürckert, B. Hollunder, A. Laux, W. Nutt,
M. Wójcik
Task Acquisition with a Description Logic Reasoner
17 pages

RR-95-03
Stephan Baumann, Michael Malburg, Hans-Guenther Hein, Rainer Hoch,
Thomas Kieninger, Norbert Kuhn
Document Analysis at DFKI
Part 2: Information Extraction
40 pages

RR-95-02
Majdi Ben Hadj Ali, Frank Fein, Frank Hoenes, Thorsten Jaeger,
Achim Weigel
Document Analysis at DFKI
Part 1: Image Analysis and Text Recognition
69 pages

RR-95-01
Klaus Fischer, Jörg P. Müller, Markus Pischel
Cooperative Transportation Scheduling
an application Domain for DAI
31 pages

1994

RR-94-39
Hans-Ulrich Krieger
Typed Feature Formalisms as a Common Basis for Linguistic Specification.
21 pages

RR-94-38
Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne,
Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger,
Klaus Netter, Günter Neumann, Stephan Oepen, Stephen P. Spackman.
DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling.
13 pages

RR-94-37
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 1: Overview.
54 pages

RR-94-36
Manfred Meyer
Issues in Concurrent Knowledge Engineering. Knowledge Base and Knowledge Share Evolution.
17 pages

RR-94-35
Rolf Backofen
A Complete Axiomatization of a Theory with Feature and Arity Constraints
49 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,
Günter Neumann, Hans Uszkoreit
COSMA – Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-19
Rainer Hoch
Using IR Techniques for Text Classification in Document Analysis
16 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

DFKI Technical Memos

1996

TM-96-02
Harold Boley
Knowledge Bases in the World Wide Web:
A Challenge for Logic Programming
8 pages

TM-96-01
Gerd Kamp, Holger Wache
CTL — a description Logic with expressive concrete domains
19 pages

1995

TM-95-04
Klaus Schmid
Creative Problem Solving
and
Automated Discovery
— An Analysis of Psychological and AI Research —
152 pages

TM-95-03
Andreas Abecker, Harold Boley, Knut Hinkelmann, Holger Wache,
Franz Schmalhofer
An Environment for Exploring and Validating Declarative Knowledge
11 pages

TM-95-02
Michael Sintek
FLIP: Functional-plus-Logic Programming
on an Integrated Platform
106 pages

TM-95-01
Martin Buchheit, Rüdiger Klein, Werner Nutt
Constructive Problem Solving: A Model Construction Approach towards Configuration
34 pages

1994

TM-94-05
Klaus Fischer, Jörg P. Müller, Markus Pischel
Unifying Control in a Layered Agent Architecture
27 pages

TM-94-04
Cornelia Fischer
PAntUDE – An Anti-Unification Algorithm for Expressing Refined Generalizations
22 pages

TM-94-03
Victoria Hall
Uncertainty-Valued Horn Clauses
31 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores
Text Skimming as a Part in Paper Document Understanding
14 pages

DFKI Documents

1997

D-97-01
Thomas Malik
NetGLTool Benutzeranleitung
40 Seiten

1996

D-96-07
Technical Staff
DFKI Jahresbericht 1995
55 Seiten

Note: This document is no longer available in printed form.

D-96-06
Klaus Fischer (Ed.)
Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems
63 pages

D-96-05
Martin Schaaf
Ein Framework zur Erstellung verteilter Anwendungen
94 pages

D-96-04
Franz Baader, Hans-Jürgen Bürckert, Andreas Günter, Werner Nutt (Hrsg.)
Proceedings of the Workshop on Knowledge Representation and Configuration WRKP'96
83 pages

D-96-03
Winfried Tautges
Der DESIGN-ANALYZER - Decision Support im Designprozess
75 Seiten

D-96-01
Klaus Fischer, Darius Schier
Ein Multiagentenansatz zum Lösen von Fleet-Scheduling-Problemen
Seiten

1995

D-95-12
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'95 Workshop:
KRDB-95 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
61 pages

D-95-11
Stephan Busemann, Iris Merget
Eine Untersuchung kommerzieller Terminverwaltungssoftware im Hinblick auf die Kopplung mit natürlichsprachlichen Systemen
32 Seiten

D-95-10
Volker Ehresmann
Integration ressourcen-orientierter Techniken in das wissensbasierte Konfigurierungssystem TOOCON
108 Seiten

D-95-09
Antonio Krüger
PROXIMA: Ein System zur Generierung graphischer Abstraktionen
120 Seiten

D-95-08
Technical Staff
DFKI Jahresbericht 1994
63 Seiten

Note: This document is no longer available in printed form.

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 pages

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-04
Victoria Hall
Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache
56 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ACCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
PuK-95, Beiträge zum 9. Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-94-02
Markus Steffens
Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

D-94-01
Josua Boon (Ed.)
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

NetGLTool Benutzeranleitung

Thomas Malik

D-97-01
Document