KfK 3558
Juli 1983

Der Graphische Editor E82

Benutzungsanleitung

G. Enderle, K.-H. Bechler, C. Fischer, U. Marek, W. Olbrich
Institut für Reaktorentwicklung

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Reaktorentwicklung

KfK 3558

Der Graphische Editor E82

Benutzungsanleitung

G. Enderle
K.-H. Bechler
C. Fischer
U. Marek
W. Olbrich

Kernforschungszentrum Karlsruhe GmbH., Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
ISSN 0303-4003

Der Graphische Editor E82

Benutzungsanleitung

Zusammenfassung:

Das Programm E82 ist ein interaktiver graphischer Editor zum Erzeugen, Ändern und Darstellen von Bildern. Er basiert auf dem genormten Graphischen Kernsystem GKS und ist in der Programmiersprache FORTRAN 77 geschrieben. Neben Fähigkeiten zum Editieren von Bildern enthält der Editor einen Präsentationsgraphik-Modul zum Erzeugen graphischer Darstellungen aus Wertefeldern. Die Benutzerkommunikation erfolgt über einen Kommandointerpretierer, der Benutzerkommandos, Makros und graphische Interaktionen bearbeitet. Dieser Bericht enthält die Benutzeranleitung für den Editor E82.

The Picture Editor E82

User Reference Manual

Summary:

The program E82 is an interactive picture-editor for creating, manipulating and displaying pictures. It is based on the standardized Graphical Kernel System and is coded in the programming language FORTRAN 77. Besides general picture editing capabilities, the editor E82 contains a presentation graphics module for generating charts from given data values. The user communication is controlled by a command processor that handles commands, macros and graphical interactions. This report contains the user guide for the editor E82.

Inhalt Seite

1. EINLEITUNG

Dieser Bericht enthält die Dokumenation für den Editor E82. Sie umfaßt die Beschreibung der Benutzerschnittstellen und des Funktionsumfangs der einzelnen Systemteile sowie den Entwurf einer Systemrealisation. Dazu gehört die Abbildung von Systemfunktionen auf GKS-Funktionen, die Programmstruktur, die Datenstruktur und die Dateistruktur. Der Editor wird in der Programmiersprache FORTRAN 77 geschrieben. Die Datenstrukturen werden daher durch FORTRAN-Common-Blöcke realisiert, die Dateien sind formatierte sequentielle FORTRAN-Files.

1.1 Grundelemente des E82

Der graphische Editor E82 ist ein Programm zur interaktiven Erstellung, Bearbeitung und Verwaltung von Bildern. Er basiert auf dem genormten Graphischen Kernsystem GKS /1/, ist in der Programmiersprache FORTRAN 77 geschrieben und somit weitgehend portabel.

Der Editor E82 besteht aus drei wesentlichen Moduln:

- dem Modul für die Bildmanipulation. Bilder können erzeugt und verändert, Teilbilder transformiert oder gelöscht werden, Darstellungsattribute wie Linienarten, Text-Zeichensätze oder Farben geändert oder Bilder gespeichert und wiedereingelesen werden.

- dem Modul für die Datenpräsentation. Er enthält Funktionen zur graphischen Darstellung von in numerischen Dateien gespeicherten Wertefeldern. Die vom Datenpräsentationsmodul erzeugten Bilder können anschließend wie auf anderem Wege erzeugte Bilder weiter behandelt werden.

- dem Modul für die Benutzerkommunikation. Er sorgt für die Kommunikation mit dem Bediener eines interaktiven Arbeitsplatzes. Im Stapelbetrieb dient er der Verarbeitung von Kommandos, die in einer Eingabedatei bereitgestellt werden. Der Modul für die Benutzerkommunikation kann auch unabhängig vom Editor E82 verwendet werden.

Alle Moduln bauen zur Darstellung von Bildern und zur Kommunikation mit dem Bediener auf GKS-Funktionen auf (siehe Abbildung 1).

Der E82 wird gegenwärtig im Institut für Reaktorentwicklung des Kernforschungszentrums Karlsruhe in Zusammenarbeit mit dem Ingenieurbüro Feldle München entwickelt.

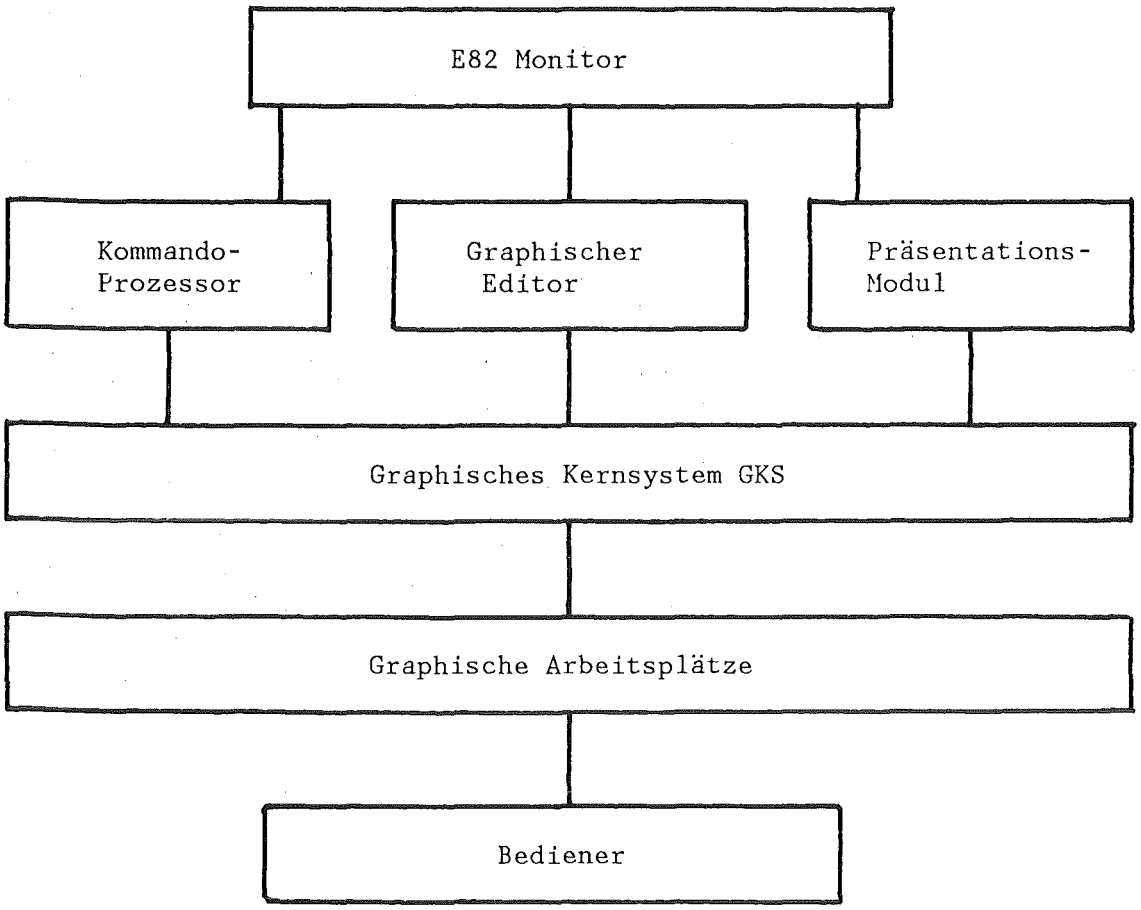


Abbildung 1: Hauptmodule des Editors E82

1.2 Was ist ein graphischer Editor?

Ein graphischer Editor dient dazu, mit Hilfe eines Rechners und eines graphischen Arbeitsplatzes graphische Darstellungen (Bilder) zu erzeugen, zu verändern und zu speichern. Die Verbindung vom Editor zum graphischen Arbeitsplatz erfolgt über ein graphisches System (in unserem Falle das GKS), das die elementare Ausgabe graphischer Daten auf den Arbeitsplatz und die Eingabe vom Arbeitsplatz verwaltet. Das graphische System ist auch für die Bildspeicherung verantwortlich.

Ein graphischer Arbeitsplatz besteht aus einem graphischen Ausgabegerät zur Darstellung der Bilder und aus Eingabegeräten, mit denen der Bediener den Editor steuert. Das Ausgabegerät ist in den meisten Fällen ein schwarz-weißer oder farbiger Bildschirm. Zur Ausgabe auf Papier, Folie oder Film kann ein Plotter (Zeichenmaschine) oder eine Kamera angeschlossen werden.

Über die Eingabegeräte wirkt der Bediener auf den graphischen Arbeitsplatz ein, der dann Eingabewerte an das graphische System und letztlich an den graphischen Editor zurückgibt. Ihrer Funktion nach denen die Eingabegeräte zum Eingeben von Texten (Text-Eingabegerät), einzelnen Koordinaten (Lokalisierer-Eingabegerät) oder Folgen von Koordinatenwerten (Strichgeber-Eingabegerät), von reellen Werten (Wertgeber-Eingabegerät), zur Auswahl einer aus einer Reihe von Möglichkeiten (Auswähler-Eingabegerät) oder zum Identifizieren von

Teilbildern auf dem Bildschirm (Picker-Eingabegerät). Die physische Realisierung dieser Eingabefunktionen auf verschiedenen grapischen Arbeitsplätzen kann sehr verschieden sein. Z.B. kann ein Koordinatenwert mit dem Steuerknüppel oder mit einem Stift auf einem Tablett oder numerisch über die alphanumerische Tastatur eingegeben werden. Die Auswähler-Funktion wird oft durch Menüs auf dem Bildschirm oder auf einem Tablett oder durch Funktionstasten realisiert. Der Editor muß so flexibel sein, daß die auf verschiedenartigen Arbeitsplätzen vorhandenen Eingabegeräte optimal genutzt werden können.

Die wichtigsten Grundfunktionen des Editors E82 zum Erzeugen, Speichern und Manipulieren von Bildern sind:

- Erzeugen von Bildern aus den GKS-Grundelementen (Linien, Marken, Texte, Flächen, Muster),

- Ändern der Darstellungsattribute der Bildelemente (u.a. Linienart, Strichstärke, Zeichenhöhe, -breite, -abstand, -richtung, Zeichensatz, Flächenmuster, Farbe),

- Löschen von Teilbildern,

- Verschieben, Drehen oder Skalieren von Teilbildern,

- Duplizieren von Teilbildern,

- Ändern der Abbildungstransformationen (z.B. Zoomen oder Schwenken),

- Einlesen und Darstellen von GKS-Bilddateien. Auf GKS-Bilddateien können Bilder langfristig gespeichert werden. Sie sind als sequentielle FORTRAN-Files realisiert;

- Ausgeben von Bildern auf eine GKS-Bilddatei,

- Einfügen von Bildern von einer GKS-Bilddatei in das aktuelle Bild,

- Umwandeln von GKS-Bilddateien in AGF-Plotfiles und zurück. Das AGF-Plotfile-Format wurde 1979 innerhalb der Arbeitsgemeinschaft der Großforschungszentren (AGF) zur Bildspeicherung und zur Bildübermittlung entwickelt und ist in breitem Umfang im Einsatz.

1.3 Präsentationsgraphik

Die Präsentationsgraphik befaßt sich mit der graphischen Visualisierung von Daten. Die Daten können aus Messungen, Rechnungen oder anderen Datenhaltungen (z.B. aus der betrieblichen Buchhaltung) stammen. Teilgebiete der Präsentationsgraphik sind die Bürographik und die Managementgraphik. Ein anderer wichtiger Bereich ist die Datenpräsentation in Wissenschaft und Technik. Der Editor E82 bietet eine Reihe von Standardformaten zur Datenpräsentation, sowohl für ein- und zwei- als auch für dreidimensionale Daten. Die wichtigsten Standardformate sind: Histogramme, Kreisdiagramme, Kurvendiagramme, Textvorlagen, Höhenlinien, Isoflächen, Reliefdarstellungen und 3D-Histogramme.

1.4 Das Graphische Kernsystem GKS

Der Editor E82 benutzt als graphisches System das Graphische Kernsystem GKS, ISO-Norm 7942, DIN-Norm 66252. Das in GKS enthaltene Konzept der Ansteuerung und Verwaltung verschiedener graphischer Arbeitplätze über eine einheitliche Geräteschnittstelle macht den Editor E82 unabhängig von den verwendeten graphischen Geräten. Der Editor E82 ist vollständig in Standard FORTRAN 77 programmiert und damit auf allen Rechnern verwendbar, die GKS und FORTRAN 77 bereitstellen. Die mit dem Editor E82 erstellten und verwalteten Bilder können langfristig auf der GKS-Bilddatei gespeichert und an andere GKS-Anwender übermittelt werden. Bilder fremden Ursprungs können als GKS-Bilddatei vom E82 übernommen und weiterverarbeitet werden.

Das GKS ist die erste internationale Norm auf dem Gebiet der graphischen Datenverarbeitung (Computer-Graphik). Diese Norm definiert nicht nur eine einheitliche Schnittstelle zwischen Anwenderprogrammen und einem graphischen System, sondern sie ist darüberhinaus die Grundlage einer einheitlichen Systematik für graphische Systeme und die Konzepte, auf denen sie aufbauen. GKS beschreibt einen Satz von Funktionen zur Programmierung graphischer Systeme. Es stellt die grundlegenden Fähigkeiten zur Erzeugung Computer-generierter zweidimensionaler graphischer Darstellungen zur Verfügung. Es unterstützt die interaktive graphische Datenverarbeitung durch Funktionen zur Dialogsteuerung und zur Werteeingabe durch einen Bediener, zur Bildstrukturierung und zur Manipulation von Teilbildern. Der GKS-Norm-Entwurf definiert den Kern eines graphischen Systems unabhängig von einer Programmiersprache. Zur Anwendung des Graphischen Kernsystems muß dieser Kern in die jeweils gewünschte Programmiersprache (wie in eine Schale) eingebettet werden, wobei die Konventionen dieser Programmiersprache beachtet werden müssen. In unserem Falle wenden wir das GKS in der Programmiersprache FORTRAN 77 an. Dabei wird die vom DIN erarbeitete GKS-FORTRAN-Schnittstelle benutzt (siehe Kapitel 11).

Die GKS-Funktionen gehören zu den Teilbereichen Darstellungselemente, Attribute, Graphische Arbeitsplätze, Transformationen, Bildstruktur, graphische Eingabe, Bilddatei und Erfrage-Funktionen. Die Funktionen sind in Leistungsstufen gegliedert, die für GKS-Realisierungen festgelegt wurden. Fehlerbedingungen werden durch eine Fehlerbehandlung abgehandelt.

1.4.1 Darstellungselemente

Darstellungselemente sind die elementaren Bestandteile, aus denen ein Bild aufgebaut ist. Im GKS gehören dazu die Linienelemente Polygon (eine Sequenz von Vektoren), Polymarke (eine Menge von Symbolen, die je einen Punkt markieren) und Text (eine Zeichenfolge). Besonders für Darstellungen auf Rasterbildschirm-Geräten sind die Darstellungselemente Füllgebiet und Zellmatrix geeignet. Ein Füllgebiet ist eine durch einen geschlossenen Polygonzug umrandete Fläche, die farbig ausgefüllt, mit einem Muster belegt oder schraffiert sein kann. Eine Zellmatrix ist die Verallgemeinerung eines aus Rasterpunkten verschiedener Farbe aufgebauten Rechteckes.

1.4.2 Darstellungsattribute

Ein Darstellungselement ist einerseits durch seine geometrische Form, andererseits durch die Art gekennzeichnet, wie es auf der Darstellungsfläche eines Gerätes optisch in Erscheinung tritt. Dies wird durch einen Satz von Darstellungsattributen beschrieben. Für die Linienelemente gehören dazu Farbe, Strichstärke, Linienart und für Polymarken die Farbe und die Symbolnummer. Texte besitzen Attribute, die die Größe, die Richtung, den Zeichenabstand, den Zeichensatz und die farbe der Zeichen beschreiben. Füllgebiet-Attribute sind Ausfüllungsart(ungefüllt, farbig ausgefüllt, gemustert, schraffiert) und zusätzlich die Musterdefinition, die Farbe oder eine Schraffurkennzeichnung.

1.4.3 Graphische Arbeitsplätze

Ein wesentliches Element der GKS-Systemarchitektur ist das Konzept eines abstrakten Graphischen Arbeitsplatzes. Dieses Konzept erlaubt die geräteunabhängige Programmierung graphischer Anwendungsprogramme.
Ein Graphischer Arbeitsplatz ist eine Verallgemeinerung realer graphischer Geräte, er enthält eine Darstellungsfläche und eine Reihe von Eingabegeräten. Ausgabe kann nacheinander oder parallel auf verschiedene Graphische Arbeitsplätze erfolgen, ebenso kann von mehreren Arbeitsplätzen Eingabe gelesen werden. Ein Darstellungselement kann, abhängig von den speziellen Fähigkeiten der Arbeitplätze, auf verschiedenen Arbeitsplätzen mit unterschiedlichen Darstellungsattributen dargestellt werden. Durch Erfragefunktionen kann das Anwenderprogramm die Eigenschaften der angeschlossenen Arbeitsplätze erfragen.

1.4.4 Transformationen

Die Plazierung der in verschiedenen Anwenderkoordinatensystemen erzeugten Graphischen Grundelemente auf einer normalisierten Gerätedarstellungsfläche und die Abbildung dieser normalisierten auf die realen Darstellungsflächen verschiedener Graphischer Arbeitsplätze mit unterschiedlichen Gerätekoordinatensystemen wird vom GKS verwaltet.
Sie kann vom Anwenderprogramm gesteuert werden.

1.4.5 Bildstruktur

Interaktive Anwendungen erfordern die Möglichkeit, Teilbilder zu manipulieren. Im GKS kann daher ein Bild in Teilbilder, "Segmente", gegliedert sein. Segmente können unabhängig voneinander transformiert, gelöscht, zu einem Graphischen Arbeitsplatz übertragen, in andere Segmente eingefügt oder durch den Bediener eines Arbeitsplatzes (beispielsweise durch Zeigen mit einem Lichtgriffel) identifiziert werden. Das GKS verwaltet Segmente in einem eigenen Segmentspeicher.

1.4.6 Eingabe

Ein Anwenderprogramm kann Werte von einem Bediener eines Graphischen Arbeitsplatzes anfordern. Das GKS kennt die sechs Eingabegeräteklassen Lokalisierer, Strichgeber, Wertgeber, Picker, Auswähler und Text. Die flexible Kontrolle dieser Eingabegeräte durch das Anwenderprogramm bietet zusammen mit der Bildsegmentierung die Grundlage für die Gestaltung interaktiver Anwendungen, ohne daß die Geräteunabhängigkeit aufgegeben werden muß.

1.4.7 Bilddatei

GKS enthält eine Schnittstelle zu einer Bilddatei, die sowohl das langfristige Speichern als auch das Wiedereinlesen von Bildern erlaubt. Die Bilddatei kann z.B. zum Archivieren von Bildern, zum Übertragen von Bildern oder zum Wiederaufnehmen einer abgebrochenen Sitzung dienen.

1.4.8 Erfragefunktionen

Der Zustand des GKS ist durch die Werte einer Reihe von Zustandslisten gegeben, die dem GKS, den angeschlossenen Graphischen Arbeitsplätzen und den Segmenten zugeordnet sind. Die Datenstruktur dieser Listen mit den Datentypen, der Bedeutung und der Vorbelegung der in ihnen enthaltenen Werte wird in einem besonderen Kapitel der Norm festgelegt. Alle in den Zustandslisten vorhandenen Werte können vom Anwenderprogramm durch Erfragefunktionen gewonnen werden.

1.4.9 Fehlerbehandlung

Falls während der Ausführung einer GKS-Funktion ein Fehler auftritt, der von der Funktion festgestellt werden kann, geht das GKS in einen besonderen Fehler-Zustand über, und die GKS-Fehlerbehandlung wird aufgerufen. Das Anwenderprogramm hat die Möglichkeit, die Fehlerbehandlung selbst zu steuern.

1.5 Computer und Geräte für die erste Version des E82

Der Editor wird zunächst auf IBM- und Siemens-Rechnerns unter dem Betriebssystem MVS/TSO entwickelt. Später soll er auf VAX- und Prime-Anlagen übertragen werden. Auf lange Sicht ist auch an die Implementierung auf einem Arbeitsplatzrechner geplant (Personal Workstation). Die Vorteile einer Entwicklung auf einem Großrechner sind das relativ gute Angebot an Hauptspeicher und Rechenzeit und die unproblematische Benutzung des vorhandenen Timesharingsystems. Nachteilig ist der nicht immer einfache Anschluß von Graphik-Arbeitsplätzen an einen Großrechner und die langsame serielle Schnittstelle zwischen Rechner und graphischem Arbeitplatz.

Als Arbeitsplätze werden während der Entwicklungsphase ein Farbrastergerät und eine Speicherröhre unterstützt.

Das AED 512 Farbrasterterminal hat eine Auflösung von 512x512 Pixeln. 256 Farben können gleichzeitig auf dem Schirm dargestellt werden, wobei jede Farbe aus einer Palette von mehr als 16 Millionen Möglichkeiten gewählt werden kann. An das Gerät ist ein Farb-Ausgabegerät angeschlossen, das das Rasterbild auf Kleinbildfilm oder auf Polaroidpapier und Polaroidfolie verschiedener Größe kopieren kann. Vor allem für die Datenpräsentation ist das Farbgerät gut geeignet.

Die Tektronix-Speicherröhre ist einfarbig und hat eine Auflösung von etwa 4000 x 3700 adressierbaren Punkten. Das Bild wird in der Phosphorschicht gespeichert, es ist absolut flimmerfrei. Allerdings ist es nicht sehr kontrastreich. Soll ein Teil des Bildes gelöscht werden, muß das ganze Bild gelöscht und der Rest des Bildes neu generiert werden, was sehr zeitaufwendig sein kann.

Zur Ausgabe von Bildern auf Plottern wird das Bild zunächst auf eine GKS-Bilddatei ausgegeben und von dort dann durch ein Stapelprogramm auf einen der an die Zentralrechner angeschlossenen Plotter gebracht.

1.6 Interaktive Benutzerführung

Integraler Bestandteil des Editors E82 ist ein Modul zur interaktiven Benutzerkommunikation mit dem Anwender. Der Bediener des Editors steuert diesen zunächst über Kommandos, die über die Tastatur eingegeben werden. Alle Kommando-Definitionen sind in Tabellen gespeichert, dadurch können Kommandos und graphische Interaktionen leicht geändert werden. Es gibt jedoch Möglichkeiten, einen Teil der Kommandos oder alle vor dem Bediener zu verstecken, indem ein Kommando oder eine Sequenz von Kommandos mit einer GKS-Auswählerfunktion assoziiert wird. So kann der Editor z.B. über eine Reihe von Funktionstasten oder über Menüfelder auf einem Tablett gesteuert werden. Es ist auch möglich, Kommandos zu Makros zusammenzufassen, um so häufig verwendete Kommandosequenzen abzukürzen. Ebenso wie die Kommandos selbst sind auch die Funktionstasten- oder Menüfeldbelegungen und die Makrodefinitionen in Tabellen gespeichert, die einem Bediener zugeordnet sind. So kann sich jeder Anwender des Editors seine eigene Anwendungsumgebung schaffen, oder eine für ihn geeignete übernehmen.

Diejenigen Daten, die der Bediener nicht im Kommando angegeben hat, werden vom Benutzer über GKS-Eingabefunktionen nachgefordert. Dabei werden eine Reihe verschiedener Prompt- und Echoarten benutzt. Prompts sind Aufforderungen an den Bediener, eine bestimmte Eingabe vorzunehmen. Echos geben dem Bediener eine Rückmeldung über seine Aktionen. Z.B. zeigt die Positionierungsmarke auf dem Bildschirm stets die durch den Steuerknüppel angesteuerte Position. Kommandos können auch in Kommandodateien abgelegt und von dort aufgerufen werden. Wird eine Aufgabenstellung dem Editor komplett über eine Kommandodatei angeboten, so kann er auch im Stapelbetrieb eingesetzt werden.

2. BENUTZERANLEITUNG

In diesem Kapitel wird die Anwendung des Editors E82 beschrieben. Nach einer einführenden Darstellung werden die einzelnen Kommandos aufgeführt und ihre Syntax und Bedeutung erläutert. Beispiele sollen die Anwendung der Kommandos verdeutlichen. Diese Benutzeranleitung enthält auch solche Kommandos, die zur Zeit der Erstellung dieses Berichts noch nicht funktionsfähig sind (noch nicht programmiert, noch nicht vollständig programmiert oder noch nicht getestet). Insofern ist diese Anleitung auch Teil der Spezifikation der Editorfunktionen.

2.1 Die Bedienung des E82

Die Bedienung des E82 erfordert Kenntnisse in folgenden Bereichen:
- Aufruf des E82 aus MVS/TSO mit der Erläuterung der erforderlichen Dateien,
- Steuerung des E82 über Kommandos und GKS-Eingabefunktionen,
- Aufbau und Inhalt der Arbeitsumgebung eines Bedieners (Workspaces und Dateien).

In diesem Abschnitt wird in diese Teilbereiche eingeführt, ohne auf spezielle Kommandos schon einzugehen.

2.1.1 Die Steuerung des E82 über Kommmandos

Das elementare Mittel zur Übermittlung einer Aufgabe an den E82 ist ein Kommando. Ein Kommando beginnt stets mit dem Kommandonamen. Dieser besteht aus bis zu 32 Zeichen, ist aus Buchstaben, Ziffern und den Zeichen '_', "#", '$', '@' aufgebaut, wobei das erste Zeichen ein Buchstabe oder '#', '$', '@' sein muß. Bei den Kommandonamen und den in den Kommandos vorkommenden Schlüsselwörtern können überall Groß- und Kleinbuchstaben wahlweise verwendet werden. Diejenigen Kommandos, die dem Kommandointerpretierer zugeordnet sind, haben vor dem Kommandonamen das Zeichen '%' stehen. Beispiele für Kommandonamen: POLYLINE, HISTO. Beispiele für Kommandonamen des Kommandointerpretierers: %TRACE, %ALLOC.

Beispiel für die Anwendung eines Kommandos:
Der Bediener gibt folgendes Kommando auf der Tastatur des graphischen Arbeitsplatzes ein (das Zeichen ↑ bedeutet die ENTER- bzw. die RETURN-Taste):

```
POLYLINE (2,3) (5,7) ↑
```

Der Editor analysiert das Kommando, stellt fest, daß er es kennt und daß es richtig und vollständig ist und veranlaßt das GKS, eine Linie von (2,3) nach (5,7) auf dem Bildschirm darzustellen (das verwendete Koordinatensystem wird später erläutert).

Das Kommando:

```
HISTO FILE23 10 ↑
```

verlangt die Erzeugung eines Histogramms, also einer Standarddarstellung der Präsentationsgraphik. Die Daten, die dargestellt werden sollen, werden aus der Datei gelesen, die der Editor unter dem Namen FILE23 kennt. Aus dieser Datei wird der zehnte Satz von Daten entnommen und als Histogramm dargestellt.

Immer dann, wenn der Editor feststellt, daß zwar der Kommandoname bekannt ist, aber das Kommando noch nicht vollständig eingegeben wurde, fordert er vom Bediener erforderliche Angaben nach. Nicht nachgefordert werden wahlfreie Angaben. Zum Nachreichen von Werten wird der Bediener durch Prompts aufgefordert, während der Eingabe zeigt ein Echo den aktuellen Stand des Eingabewertes. Beispiele:

POLYLINE ↑

Hier fehlen erforderliche Angaben, nämlich die Koordinatenwerte des Polygonzuges. Daher erscheint jetzt das Positionierungskreuz auf dem Bildschirm (als Prompt). Der Bediener kann nun den Steuerknüppel betätigen. Das Positionierungskreuz folgt seinen Bewegungen auf dem Bildschirm (Echo) und zeigt die aktuelle Position. Immer wenn der Bediener eine Taste drückt, wird der Koordinatenwert an den Editor übermittelt, der ihn zum Aufbau des Polygonzuges benutzt. Durch eine spezielle Taste (bei uns die Q-(Quit) oder E-(Ende)-Taste) wird der Eingabevorgang abgebrochen, der bei Drücken dieser Taste anstehende Koordinatenwert wird nicht mehr übernommen.

HISTO FILE23 ↑

Hier fehlt die Nummer des Datensatzes im Vergleich zu obiger Eingabe, jedoch ist diese Angabe wahlfrei. Beim Fehlen der Nummer nimmt der Editor den nächsten Satz, den er in der Datei FILE23 findet. Daher wird in diesem Fall kein Wert vom Bediener nachgefordert.

Wie ein Kommando strukturiert ist, d.h. welche Angaben in welcher Reihenfolge angegeben werden müssen oder können, wird in den folgenden Abschnitten beschrieben (Syntax der Kommandos). Ebenso wird die Wirkung der Kommandos beschrieben (Semantik der Kommandos). Der Editor E82 hat die Syntax der Kommandos in Tabellen gespeichert, die von Dateien eingelesen werden. Es ist so leicht möglich, neue Kommandos einzufügen, ohne das Programm zu ändern. Auch der Übergang von englischen zu deutschen Kommandonamen und Schlüsselwörtern ist so leicht möglich.

In den Kommandos kommen folgende Arten von Angaben vor:

- der Kommandoname. Beispiele: POLYLINE, HISTO, %TRACE. Der Kommandoname kann abgekürzt werden, solange der eingegebene Teil noch eindeutig ist. TEXT kann so zu TE abgekürzt werden, aber POLYLINE nicht zu PO, sondern allenfalls zu POLYL, da es ein weiteres Kommando POLYMARKER gibt. Dieses kann zu POLYM abgekürzt werden.

- Schlüsselwörter (Benennungen). Sie werden wie Kommandonamen gebildet, d.h. sie bestehen aus Buchstaben und Ziffern und den oben angeführten Sonderzeichen. In Benennungen sind Groß- und Kleinbuchstaben gleichbedeutend. Beispiele: FILE, file, P1. Ebenso wie Kommandonamen können Schlüsselwörter abgekürzt werden, solange sie noch eindeutig bleiben.

- Zeichenfolgen. Sie können aus beliebigen Folgen von Zeichen bestehen, die in Apostrophe ('...') eingeschlossen sind. Innerhalb einer Zeichenfolge wird das Apostroph-Zeichen selbst durch eine Wiederholung dargestellt. Innerhalb von Zeichenketten sind Groß- und Kleinbuchstaben unterschiedliche Zeichen. Beispiele: '1 1 0', 'eine Zeichenkette', 'DON''T CARE'.

- Ganze Zahlen. Beispiele: 4, 123, 007.

- Reelle Zahlen. Gültig ist die von FORTRAN bekannte Notation für REAL-Zahlen einfacher Genauigkeit. Vor und hinter dem Dezimalpunkt muß mindestens eine Ziffer stehen. Wo eine reelle Zahl erforderlich ist, ist stets auch eine ganze Zahl gültig. Beispiele: 3.415, 7.3E-5, 1.0E10.

- Koordinatenwerte. Sie bestehen aus zwei reellen Zahlen, die durch Komma getrennt und in Klammern eingeschlossen werden. Beispiele: (3,4), (5.2,4.8), (1 , 2.0E2).

- Segmentnamen. Sie bezeichnen Teilbilder. Im interaktiven Betrieb werden Segmente meist durch Picken (d.h. durch das Anfahren mit dem Fadenkreuz auf dem Bildschirm) identifiziert, aber in Kommandodateien können auch konstante Segmentnamen sinnvoll sein. Segmente bestehen aus der Zeichenfolge 'S%' und einer ganzen Zahl. Beispiele: S%1, S%16000, S%004.

2.1.2 Verwendung von Variablen in Kommmandos

Vor allem in Kommandodateien, aber auch im interaktiven Betrieb, kann es sinnvoll sein, statt der konstanten Werte im Kommando Variable zu benutzen, denen unterschiedliche Werte zugewiesen werden können.
Die im folgenden Beispiel gegebenen Kommandos sollen in einer Kommandodatei stehen (dabei werden die Kommandos durch Semikolon ';' abgeschlossen):

```
%DECLARE %K1,%K2,%K3,%K4 LOCATOR;
%SET %K1 = ;
%SET %K2 = %K1+(2,0);
%SET %K3 = %K2+(0,1);
%SET %K4 = %K3-(2,0);
AREA %K1 %K2 %K3 %K4;
```

Durch das erste Kommando werden vier Variable %K1, %K2, %K3 und %K4 als LOCATOR vereinbart, es handelt sich also um Variable, deren Wert stets ein Koordinatenwert sein muß. Alle Variablennamen beginnen mit dem Zeichen '%', danach werden sie wie Benennungen gebildet. Das zweite Kommando weist der Variablen %K1 einen Wert zu. Da der Wert im Kommando erforderlich und nicht im Kommando angegeben ist, wird der Editor den Wert vom Bediener über den Steuerknüppel nachfordern. %K1 erhält dann den vom Bediener eingegebenen Koordinatenwert. Die folgenden Kommandos weisen den Variablen %K2, %K3 und %K4 Werte zu, die aus %K1 gebildet werden, so daß sie die Ecken eines 2 x 1 Einheiten großen Rechteckes bilden, mit der linken unteren Ecke an der vom Bediener eingegebenen Stelle. Im letzten Kommando werden die vier Koordinatenwerte zum Erzeugen einer gefüllten Fläche benutzt.

Entsprechend den Datentypen für die Konstanten eines Kommandos können auch Variable folgende Datentypen haben:

- Benennungen (NAME),
- Zeichenfolgen (STRING),
- Ganze Zahlen (INTEGER),
- Reelle Zahlen (REAL),
- Koordinatenwerte (LOCATOR),
- Segmentnamen (SEGMENT).

Variable sind normalerweise nur in der Kommandoquelle bekannt, in der sie deklariert sind, also z.B. in einer Kommandodatei oder in einer Makrodefinition. Ihr Geltungsbereich kann aber (durch die Angabe GLOBAL in der Deklaration) auf untergeordnete Kommandoquellen (also gerufene Makros oder aufgerufene weitere Kommandodateien) erweitert werden.

2.1.3 Funktionstasten und Makros

Um oft vorkommende Kommandosequenzen nicht immer jedesmal wiederholen zu müssen, kann man sie zusammenfassen und mit einem Kurznamen versehen. Dieser Kurzname kann nun wie ein Kommandoname verwendet werden. Wird er aufgerufen, wird die gleiche Wirkung wie beim Aufruf der Sequenz von Kommandos erzielt. Über Makroparameter kann das Verhalten des Makros beim Aufruf noch beeinflußt werden. Das folgende Beispiel definiert ein Makro zum Erzeugen eines gefüllten Rechteckes. Der linke untere und der rechte obere Punkt werden über Makroparameter übergeben:

```
%MACRO REC ↑
   %DECLARE %M1,%M2 LOCATOR MPARM ↑
   %DECLARE %K1,%K2 LOCATOR ↑
   %SET %K1 = (%M2.X,%M1.Y) ↑
   %SET %K2 = (%M1.X,%M2.Y) ↑
   AREA %M1 %K1 %M2 %K2 ↑
%ENDMACRO ↑
```

Die als MPARM deklarierten Variablen %M1 und %M2 erhalten beim Aufruf des Makros einen Wert durch positionelle Zuordnung. Der zuerst angegebene Wert wird dem zuerst deklarierten Makroparameter zugewiesen, usw. Beispiel:

```
REC (3,3) (10,5) ↑
```

Der erste Makroparameter %M1 erhält den Wert (3,3), der zweite, %M2, den Wert (10,5). Durch diesen Makroaufruf wird also ein gefülltes Rechteck mit den Ecken bei (3,3) und (10,5) erzeugt.

```
REC ↑
```

In diesem Fall fehlt die Angabe der Parameter beim Aufruf. Wie bei einem normalen Kommando werden sie daher vom Bediener nachgefordert und anschließend den Variablen %M1 und %M2 zugewiesen.

Jedes Kommando und jedes Makro kann mit einer Auswählereingabe (Funktionstaste oder Menüfeld) assoziiert werden, so daß bei der Betätigung des Auswählers das Kommando oder das Makro ausgeführt wird. Beispiel:

```
%FKEY 5 REC ↑
```

Damit wird der Funktionstaste Nr. 5 das REC-Makro zugeordnet. Beim Drücken der Funktionstaste wird das REC-Makro aufgerufen, die beiden Koordinatenwerte vom Bediener angefordert und ein gefülltes Rechteck gezeichnet.

2.1.4 Einfügen von Kommandodateien und Arbeiten im Stapelbetrieb

Ganze Folgen von Kommandos können auf einer Datei gespeichert sein. Von dort können sie beliebig oft eingelesen und ausgeführt werden. Das Erstellen von solchen Dateien erfolgt mit den üblichen Texteditoren (z.B. SPF-EDIT oder TSO-EDIT). Das Aufrufen einer Kommandodatei erfolgt mit Hilfe des %INCLUDE-Kommandos. Beispiel:

```
%INCLUDE KOMFILE ↑
```

Damit werden die Kommandos von der Datei eingelesen, die dem Editor unter dem Namen KOMFILE bekannt ist.

2.1.5 Dateien und Workspaces

Alle Daten, die sich von Bediener zu Bediener oder von graphischem Arbeitsplatz zu graphischem Arbeitsplatz oder von Anwendungsbereich zu Anwendungsbereich ändern können, werden vom Editor E82 in Tabellen verwaltet. Diese Tabellen sind zu Einheiten zusammangefaßt, die "Workspaces" heißen. Z.B. stehen in solchen Workspaces die Standard-attribute für graphische Darstellungen, also Farben, Zeichensätze, Zeichengrößen, oder die definierten Makros und Funktionstasten-Belegungen, oder die dem Editor bekannten Dateien.

Während der Anwendung des Editors sind alle Workspaces in FORTRAN-Datenbereichen (Commons) im Arbeitsspeicher des Rechners abgelegt. Durch Bedienerkommandos (z.B. durch eine neue Makrodefinition oder eine Attributänderung) werden die Tabelleninhalte geändert und stehen danach in geänderter Form zur Verfügung. Durch besondere Kommandos ist es möglich, die Workspaces auf externe Dateien auszugegeben und von dort wieder einzulesen. So kann ein Bediener eine ganze Reihe von verschiedenen Workspaces für verschiedene Anwendungen oder verschiedene graphische Arbeitsplätze haben.

Der Editor E82 verwaltet während seiner Anwendung eine Tabelle aller Dateien, die er benötigt. Das können die Workspace-Dateien sein, die Dateien mit den Daten für die Präsentationsgraphik, oder GKS-Bild-dateien. Die Dateien werden vom Bediener über einen Dateinamen angesprochen. Dateien können durch die Editorkommandos %ALLOC, %FREE, %REWIND mit einem Namen versehen und allokiert, wieder deallokiert oder geschlossen werden.

Am Beginn einer Editor-Anwendung, nach dem Aufruf, werden die internen Tabellen mit Werten von externen Dateien gefüllt. Welche Dateien dazu benutzt werden, steht in einem Datei-Workspace, der als erstes über eine feste Filenummer eingelesen wird.

2.1.6 Trace- und Helpfunktionen

Vor allem während der Erstellung des E82 sind Hilfsroutinen zur Fehlersuche wichtig. Ein wichtiges Hilfsmittel ist der E82_Trace. Jede E82-Routine registriert nach ihrem Aufruf und vor dem Rücksprung ihren Namen und wichtige Daten auf einer Tracedatei, sofern der Trace

eingeschaltet ist. Der Trace ist standardmäßig ausgeschaltet. Er kann durch

```
%TRACE ON ↑
```

eingeschaltet und durch

```
%TRACE OFF ↑
```

wieder ausgeschaltet werden.

Die %HELP Funktion ohne Parameter gibt Erläuterungen zum Zustand des Editors. Zusammen mit einem Kommandonamen gibt sie Erläuterungen über das Kommando auf den Bildschirm aus.

2.1.7 Der Aufruf des E82

Der E82 wird aus TSO oder aus dem SPF-Menü 6 durch Aufruf der TSO-Kommandoprozedur E82 aufgerufen. Beispiele:

```
E82 ↑
```

Diese Form ist dann möglich, wenn die E82-CLIST-Datei auf den OS-DD-Namen SYSPROC allokiert ist.

```
EXEC 'TSO147.E82.CLIST(E82)'
```

Diese Form ist immer möglich. (Vorausgesetzt, die Datei TSO147.E28.CLIST ist die E82-CLIST-Datei und enthält das Member E82). Die Kommandoprozedur ist folgendermaßen aufgebaut:

```
PROC 0                                          /*Keine Parameter    */
CONTROL  MAIN NOFLUSH NOMSG                     /*Damit beim FREE    */
FREE     FILE(FT02F001)                         /* die Prozedur      */
FREE     FILE(FT06F001)                         /* nicht beendet     */
FREE     FILE(FT14F001)                         /* wird und keine    */
FREE     FILE(FT17F001)                         /* Nachrichten       */
FREE     FILE(FT20F001)                         /* erfolgen. Die     */
FREE     FILE(FT21F001)                         /* FREEs sind nur    */
FREE     ATTRLIST(XXX)                          /* zur Sicherheit.   */
CONTROL MSG                                     /*Messages wieder ein*/
ERROR GOTO LAB                                  /*Bei Fehler: FREE   */
ATTN GOTO LAB                                   /*Ebenso bei ATTN    */
ALLOC SYSOUT(N)              F(FT02F001)        /*GKS-Protokoll      */
ALLOC SYSOUT(A) DEST(RM006) F(FT06F001)         /*E82, FORTRAN-Mess. */
ALLOC SYSOUT(A) DEST(RM006) F(FT14F001)         /*E82-Trace          */
ALLOC SYSOUT(A) DEST(RM006) F(FT20F001)         /*GKS-Trace          */
ALLOC DA('TSO147.E82.DATA(FE82COM)') F(FT17F001) /*Hauptcommon   */
ALLOC DA('TSO147.E82.DATA(FECWSF)')  F(FT21F001) /*E82-Dateiliste*/
CALL     'TSO147.E82.MODS(E82)'                 /*Aufruf des E82     */
LAB: ERROR GOTO END                             /*Fehler im Fehler   */
ATTN GOTO END                                   /*Ebenso bei ATTN    */
FREE     FILE(FT02F001)                         /*Freigeben der      */
FREE     FILE(FT06F001)                         /* Dateien           */
FREE     FILE(FT14F001)
FREE     FILE(FT17F001)
FREE     FILE(FT20F001)
FREE     FILE(FT21F001)
FREE     ATTRLIST(XXX)
END: END                                        /*Ende               */
```

Die SYSOUT-Dateien werden allokiert, weil das mit dem FORTRAN-OPEN-Statement im E82 nicht möglich ist. Der Hauptworkspace des E82 und die Dateiliste werden vom E82 nach dem Aufruf über feste Filenummern (17 und 21) eingelesen, sie müssen daher in der Kommandoprozedur allokiert sein. Nach dem Einlesen der Dateiliste werden alle anderen Dateien aufgrund der dort vorhandenen Angaben vom E82 intern selbst allokiert.

2.1.8 Die Syntax der Kommandobeschreibungen

Die für die folgenden Kommandobeschreibungen verwendete Form wird hier erläutert. Die Beispiele in diesem Abschnitt sind zwar an E82-Kommandos angelehnt, aber oft vereinfacht, die vollständige Beschreibung findet sich in den Abschnitten 2.2 bis 2.4. Es gibt vier Gruppen von Elementen in den Kommandobeschreibungen:

- Kommandonamen und Schlüsselwörter. Sie müssen so im Kommando angegeben werden, wie sie in der Beschreibung vorkommen.
- Bezeichner von Teilkommandos. Sie werden in der Beschreibung weiter erläutert.
- Bezeichner für Variable und Konstanten. An ihrer Stelle muß im Kommando eine Konstante oder eine Variable stehen. Ihr Datentyp und die Bedeutung wird in der Beschreibung angegeben.
- Symbole, die bedeuten, daß ein Teil des Kommandos wiederholt werden kann oder weggelassen werden kann.

Kommandonamen:

Kommandonamen werden mit Großbuchstaben angegeben. In der Anwendung des Kommandos dürfen dann statt Groß- auch Kleinbuchstaben verwendet werden. Außerdem darf der Name bis auf die signifikanten Buchstaben abgekürzt werden.

```
Beschreibung:                    Anwendung:
   END                              END ↑
                                    end ↑
                                    En ↑
```

Der Kommandoname ist in diesem Beispiel "END".

Schlüsselwörter:

Schlüsselwörter werden ebenfalls mit Großbuchstaben angegeben. In der Anwendung dürfen Kleinbuchstaben benutzt werden und die Schlüsselwörter dürfen abgekürzt werden.

```
Beschreibung:                    Anwendung:
   TRACE ON                         TRACE ON ↑
                                    TRACE on ↑
```

Das Schlüsselwort ist in diesem Beispiel "ON".

Bezeichner für Teilkommandos:

Sie werden in der Beschreibung mit kleinbuchstabigen Benennungen angegeben und dann weiter erläutert. Der nach den Zeichen '::=' stehende Teil der Beschreibung ersetzt den Bezeichner in der Kommandobeschreibung.

```
Beschreibung:                    Ist gleichwertig mit der Beschreibung:
   ATTRIBUTE TEXT font              ATTRIBUE TEXT FONT nummer
   font ::= FONT nummer
```

Der Bezeichner für das Teilkommando ist in diesem Falle "font".

Bezeichner für Variable und Konstanten:
Sie werden in der Beschreibung ebenfalls mit kleinbuchstabigen Benennungen angegeben und dann weiter erläutert. Die weitere Erläuterung enthält den Datentyp und die Bedeutung der Konstanten oder Variablen. Der Datentyp kann NAME, STRING, LOCATOR, INTEGER, REAL oder SEGMENT sein. Bei NAME und STRING kann außerdem eine maximale Länge angegeben sein. Ist nichts angegeben, ist die maximale Länge 8.

Beschreibung:
```
HISTO fname fnr
fname: NAME, Name der Datei mit
   den HISTO-Daten
fnr:   INTEGER, Nummer des
   Datensatzes
```
Anwendung:
```
HISTO FILE1 3 ↑
HISTO A123#1 123 ↑
HISTO %FILE %NR ↑
```
Die Bezeichner für Konstanten oder Variable sind in diesem Falle "fname" und "fnr".

Symbole für Alternativen:
Wird in der Kommandobeschreibung eine Reihe von Alternativen angegeben, so ist eine und nur eine der Alternativen bei der Anwendung des Kommandos auszuwählen. Alternativen sind zwischen den Zeichen '{' und '}' eingeschlossen und durch '|' getrennt.

Beschreibung:
```
PARMSPIE {SEGTEX | NOSEGTEX | RADIUS pr | CENPOINT p0 }
pr:    REAL, Radius des Kreises
p0:    LOCATOR, Mittelpunkt des Kreises
```

Anwendung:
```
PARMSPIE SEGTEX ↑
PARMSPIE NOSEGTEX ↑
PARMSPIE RADIUS 1.4 ↑
PARMSPIE CENPOINT (10,10) ↑
```

Symbole für optionale Kommandoteile:
Wird in der Kommandobeschreibung ein Kommandoteil zwischen '[' und ']' eingeschlossen, so bedeutet das, daß der Kommandoteil weggelassen werden kann.

Beschreibung:
```
HISTO fname [ fnr ]
fname: NAME, Name der Datei mit den HISTO-Daten
fnr:   INTEGER, Nummer des Datensatzes. Falls die Nummer fehlt,
   wird der nächste Datensatz auf der Datei dargestellt.
```

Anwendung:
```
HISTO FILE1 3 ↑
HISTO A123#1 ↑
HISTO %FILE ↑
```

Symbole für Wiederholungen:
Wird in der Kommandobeschreibung ein Kommandoteil zwischen '[' und ']*' eingeschlossen, so bedeutet das, daß der Kommandoteil einmal oder beliebig oft mal angegeben oder weggelassen werden kann. Werden die Werte in der Wiederholung angefordert, so kann die Wiederholung bei der Eingabe über Fadenkreuz durch Drücken der E- oder Q-Taste abgebrochen werden. Bei der Eingabe über die Tastatur erfolgt der Abbruch durch Eingabe eines Leerstrings.

Beschreibung:

```
POLYLINE [ punkt ]*
punkt: LOCATOR, Punkt des Polygonzuges
```

Anwendung:

```
POLYLINE
POLYLINE (1,1) (3,3) ↑
POLYLINE (1,1) (10,1) (10,10) (1,10) (1,1) ↑
```

Abschluß der Kommandos:
In der Beschreibung der Kommandos ist der Abschluß nicht besonders vermerkt. Bei Eingabe eines Kommandos von der Tastatur wird ein Kommando durch die Taste "RETURN" oder "ENTER" abgeschlossen. Dies ist in den Beispielen durch die Zeichen '↑' angedeutet. Werden mehrere Kommandos in eine Zeile geschrieben, oder stehen die Kommandos in einer Datei, so sind sie durch Semikolon ';' abzuschließen.

Beispiel für eine vollständige Kommandobeschreibung:

Beschreibung:

```
CHANGE [segment]* [attribute]* [ALL]
segment:  SEGMENT, Segment, dessen Attribute geändert werden sollen.
attribute ::= { LTYPE ltype | LWIDTH lwidth | MHEIGHT mheight |
   STYLE style |  PATTERN n m [farben]* | COLOUR index }
ltype:    INTEGER, Linienart
lwidth:   REAL, Strichstärke
mheight:  REAL, Markenhöhe
style:    { HOLLOW | PATTERN | FILLED | HATCH }, Füllart
n,m:      INTEGER, Dimension der Mustermatrix
farben:   INTEGER, Mustermatrix
```

Anwendung:

```
CHANGE S%1 LTYPE 3 ↑
CHANGE S%5 S%6 S%7 COLOUR 200 STYLE PATTERN PATTERN 2 1 21 22 ↑
CHANGE %T1 %T2 COLOUR %COL MHEIGHT %H LWIDTH %W ALL ↑
```

2.2 Das Editieren von Bildern

In diesem Abschnitt werden die Grundfunktionen des Bildeditors beschrieben, die Bildeditor-Kommandos des E82 aufgelistet und ihre Parameter und ihre Wirkung erläutert.

2.2.1 Aufbau von Bildern

Die kleinsten Einheiten, aus denen ein Bild aufgebaut werden kann, sind die graphischen Grundelemente oder Darstellungselemente. Dies sind zunächst die GKS-Darstellungselemente Linien (POLYLINE), Punktesymbole oder Marken (POLYMARKER), Texte (TEXT), gefüllte Flächen (AREA) und verschiedenfarbig gefüllte Rechteckraster (CELL). Darüberhinaus stellt der Editor E82 auch die Darstellungselemente Kreis (CIRCLE), Kreisbogen (ARC), Kreissegment (ARCSEG), Kreissektor (ARCSEC) und Rechteck (RECTANGLE) zur Verfügung. Die Darstellungselemente sind durch ihre geometrische Form definiert. Diese ändert sich nicht beim Übergang auf ein anderes Gerät, sie kann lediglich durch Transformationen linear transformiert (verschoben, vergrößert, verkleinert, gedreht) werden. Darstellungselemente werden im E82 durch entsprechende Kommandos erzeugt. Z.B wird ein Linienzug durch das POLYGON-Kommando erzeugt.

Diejenigen Aspekte der Darstellung der Grundelemente, die nicht durch die geometrische Form gegeben sind, heißen Darstellungsattribute. Dazu gehören Farbe, Strichstärke, Markengröße, Zeichenhöhe und Zeichenbreite, Textrichtung, Textabstand und vieles mehr. Die Darstellungsattribute können auf unterschiedlichen Geräten unterschiedlich ausgewertet werden. Es ist z.B. einzusehen, daß auf einem Gerät ohne Farbfähigkeiten die Farbattribute einfach ignoriert werden. Bei der Erzeugung von Darstellungselementen erhalten sie im GKS und im E82 die derzeit gültigen Attribute zugewiesen. Diese Attribute werden zu Beginn der E82-Anwendung aus einer hierfür vorhandenen Workspace-Datei eingelesen. Mit Hilfe des ATTRIBUTE-Kommandos können die Attributwerte geändert werden. Die Attribute schon erzeugter Darstellungselemente können mit Hilfe des CHANGE-Kommandos geändert werden.

Ein Bild ist im GKS und im E82 in Teilbilder gegliedert, die Segmente heißen. Bei der Erzeugung von Darstellungselementen erhält zunächst jedes Element ein eigenes Segment. Es kann somit einzeln verändert (z.B. transformiert oder gelöscht) werden. Auch die Darstellungen der Präsentationsgraphik sind in Teilbilder gegliedert. Segmente werden für Manipulationen dadurch ausgewählt, daß sie mit Hilfe des Picker-Eingabegerätes identifiziert, also gepickt, werden. Das läuft z.B. so ab, daß mit dem Steuerknüppel das Positionierungssymbol auf dem Bildschirm über das zu verändernde Element bewegt und dann eine Taste der alphanumerischen Tastatur gedrückt wird. Die wichtigsten Funktionen für Segmente sind Transformationen (Kommandos ROTATE, ROTP, SCALE, SCALEP, TRANSLATE), das Löschen (DELETE) und das Kopieren (COPY). Ein Segment, das aus mehreren Darstellungselementen besteht, kann mit Hilfe des Kommandos PRIMITIVE in einzelne Segmente zerlegt werden. Eine Reihe von Segmenten kann mit Hilfe des Kommandos SEGMENT zu einem einzigen Segment zusammengefaßt werden.

Die Darstellungselemente werden in einem rechtwinkligen kartesischen Koordinatensystem erzeugt. Ein rechteckiger Teil dieses Koordinatensystems wird auf den Bildschirm abgebildet. Dieser darstellbare Teil des Koordinatensystems heißt Fenster oder Window. Das Standardfenster des E82 umfaßt den Bereich (0,0) bis (20,20), d.h. alle sichtbaren

Teile eines Bildes liegen normalerweise in diesem Koordinatenbereich. Durch das WINDOW-Kommando kann das Fenster geändert werden. Derjenige Bereich des Bildschirms, auf dem das Bild abgebildet wird, heißt Darstellungsfeld oder Viewport. Das ist normalerweise der größtmögliche quadratische Bereich, der noch auf dem Bildschirm Platz hat. Das Darstellungsfeld kann durch das VIEWPORT-Kommando geändert werden. Das Fenster wird stets genau auf den Darstellungsbereich abgebildet. Eine kontinuierliche Vergrößerung oder Verkleinerung des Fensters ergibt einen Zoom-Effekt (Kommando ZOOM), eine Verschiebung des Fensters ein Schwenken über das Bild (Kommando PAN).

Der Editor E82 verwaltet Bilder, Segmente und Grundelemente mit Hilfe von GKS-Funktionen. Ein Bild hat einen Bildnamen, der aus einem Text der Länge 8 besteht. Ein Segment hat einen Segmentnamen, der eine ganze Zahl ist. Während der Segmentname im allgemeinen für den Anwender unwichtig ist, da er die Segmente durch Picken identifiziert, sind die Bildnamen wichtig, da sie es erlauben, ein bestimmtes Bild von einer Bilddatei einzulesen, bzw. ein Bild auszugeben. Daher wird der Bildname beim Schreiben auf die Bilddatei angegeben. Während der Anwendung des E82 gibt es stets ein aktuelles Bild. Mit dem Kommando CLEAR kann das aktuelle Bild gelöscht werden. Die Taste "HERE IS" am AED-Terminal und die Taster "ERASE" am Tektronix-Terminal löschen dagegen zwar den Bildschirm, aber nicht das Bild. Das aktuelle Bild kann jederzeit mit dem RESHOW-Kommando neu generiert werden.

2.2.2 Dateien und Workspaces für das Editieren

Sollen Bilder auf einer Bilddatei gespeichert und von dort wieder eingelesen werden, so können sie über den Bildnamen eindeutig identifiziert werden, der beim Schreiben der Bilddatei angegeben wird. Ist ein Bildname auf der Bilddatei nicht vorhanden (z.B. bei nicht im E82 erzeugten Bilddateien), so kann ein Bild nur durch seine Position auf der Bilddatei identifiziert werden (das wievielte Bild auf der Bilddatei ist es?).

Das aktuelle Bild kann mit dem SAVE-Kommando auf eine externe Bilddatei abgelegt werden. Von dort kann es mit dem RESTORE-Kommando wieder eingelesen werden. Die aktuelle Bilddatei enthält stets nur ein Bild, eben dasjenige, das zuletzt mit der SAVE-Anweisung gerettet wurde. Daher ist hier auch die Angabe eines Namens nicht erforderlich. Die aktuelle Bilddatei dient der kurzfristigen Absicherung des aktuellen Bildes gegen Abstürze und unbeabsichtigte Fehlbedienungen (Der E82 sollte eigentlich nicht abstürzen, aber kein Programm ist perfekt, und das Betriebssystem soll auch schon einmal zusammengebrochen sein). Beim Abschluß des E82 wird, wenn das aktuelle Bild nicht gelöscht oder auf eine Bilddatei gespeichert wurde, das Bild auf die aktuelle Bilddatei gerettet.

Langfristig werden die Bilder auf Bilddateien gespeichert, die mehr als ein Bild in sequentieller Folge enthalten können. Die Eingabe von solchen Bilddateien und die Ausgabe des aktuellen Bildes auf eine Bilddatei ist möglich. Die Eingabe- und Ausgabebilddateien werden mit dem USE-Kommando ausgewählt. Die Kommandos READ und WRITE dienen dem Lesen und Schreiben von Bilddateien.

Das Kommando INSERT fügt ein Bild von einer Bilddatei oder das Bild von der aktuellen Bilddatei in das aktuelle Bild ein. Abbildung 2 gibt einen Überblick über Bilddateien und die zugehörigen Kommandos.

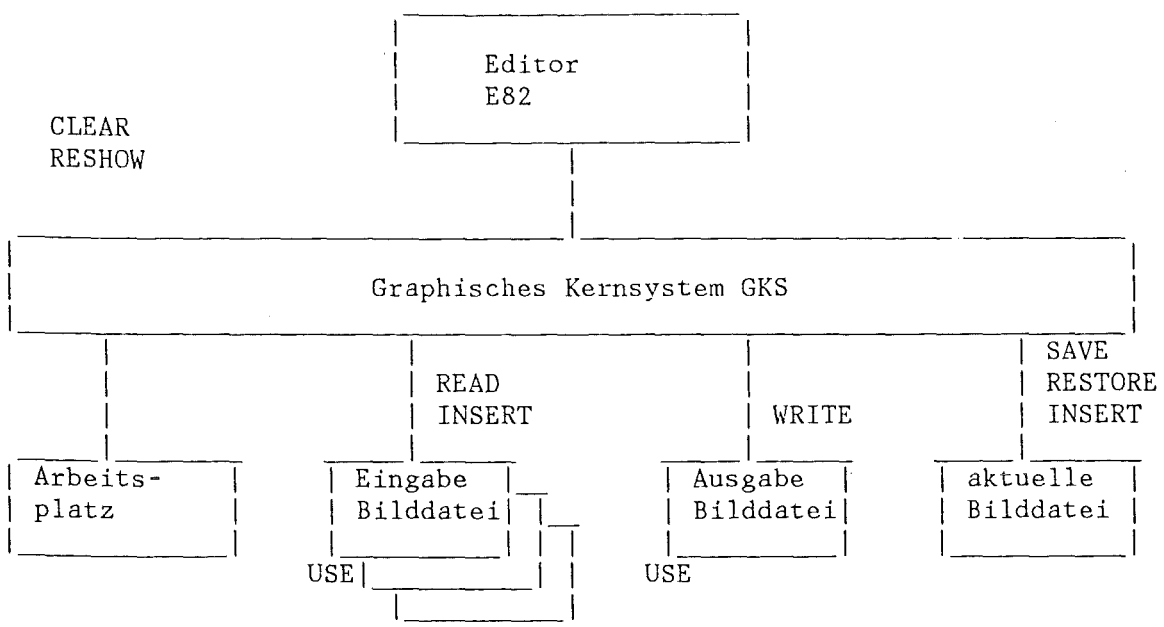


Abbildung 2: Bilddateien des Editors E82

Neben den Bilddateien ist die Datei für den Editor-Workspace für das Editieren von Bildern wichtig. In ihr werden die Parameter für den E82-Bildeditor langfristig gespeichert. Dazu gehören u.a. die schon beschriebenen Werte für Attribute, für Window und Viewport, für Eingabe- und für Ausgabebilddateien. Beim Aufruf des E82 wird der Workspace von der in der Dateiliste angegebenen Datei eingelesen. Mittels des WORKSPACE-Kommandos kann ein anderer Workspace eingelesen oder der aktuelle Workspace auf eine Datei gespeichert werden. Das INQUIRE-Kommando dient der Abfrage von Werten aus dem aktuellen Workspace.

2.2.3 Kommandos für Grundelemente

Die GKS-Grundelemente können durch Kommandos erzeugt werden. Dabei werden die gerade gültigen Attribute verwendet. Die Attribute werden durch das ATTRIBUTE-Kommando gesetzt. Sie können durch das CHANGE-Kommando nachträglich geändert werden. Alle Grundlemente werden, wenn sie erzeugt sind, zum aktuellen Grundelement. Das ist wichtig, wenn sie sofort anschließend transformiert oder gelöscht werden sollen.

POLYLINE
Das POLYLINE-Kommando erzeugt einen Polygonzug. Dabei werden die gerade gültigen POLYLINE-Attribute verwendet.

Syntax:

POLYLINE [punkt]*
punkt: LOCATOR, Punkt des Polygonzuges

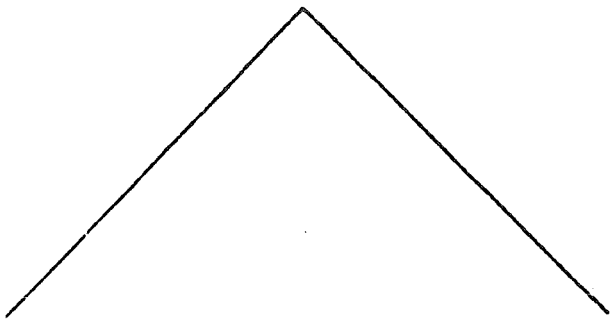
Beispiel:

POLYLINE ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren desselben ist jedes Koordinatenpaar durch das Drücken irgendeiner alphanumerischen Taste einzugeben. Ab dem zweiten Koordinatenpunkt erscheint sofort die Verbindungslinie zum letzten Punkt. Das Polygon wird abgeschlossen, wenn eine der Tasten "E" oder "Q" gedrückt wird, der dann anstehende Koordinatenwert wird nicht mehr übernommen.

POLYLINE (5,5) (10,10) (15,5) ↑

Nach der Eingabe dieses Befehls wird ein Polygonzug mit den Koordinaten :
Punkt 1 : X = 5, Y = 5
Punkt 2 : X = 10, Y = 10 und
Punkt 3 : X = 15, Y = 5 gezeichnet.



C:POLYLINE (5,5) (10,10) (15,5)

Bemerkung :

Die Farbe des Polylines läßt sich durch den ATTRIBUTE Befehl vorher festlegen oder durch CHANGE nachträglich verändern.

POLYMARKER

Das POLYMARKER-Kommando erzeugt einen Satz von Marken gleichen Typs. Dabei werden die gerade gültigen POLYMARKER-Attribute verwendet.

Syntax:

POLYMARKER [punkt]*
punkt: LOCATOR, Markenposition.

Beispiel:

POLYMARKER ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren des selben ist jedes Koordinatenpaar durch das Drücken irgendeiner alphanumerischen Taste einzugeben. Danach wird die Polymarke auf dem Bildschirm ausgegeben. Die Eingabe wird beendet, wenn eine der Tasten "E" oder "Q" gedrückt wird.

POLYMARKER (4,7) (11,3) (5,5) ↑

Nach der Eingabe dieses Befehls werden zwei Polymarken an den Koordinaten :
Punkt 1 : X = 4, Y = 7 und
Punkt 2 : X = 11, Y = 3 gezeichnet.
Punkt 3 : X = 5, Y = 5 gezeichnet.

C: POLYMARKER (4,7) (11,3) (5,5)

Bemerkung :

Die Farbe, der Typ und die Größe des Polymarkers lassen sich durch den ATTRIBUTE Befehl vorher festlegen oder durch CHANGE nachträglich verändern.

TEXT
Das TEXT-Kommando erzeugt eine Zeichenfolge. Dabei werden die gerade gültigen TEXT-Attribute verwendet.

Syntax:
TEXT string punkt
string: STRING, darzustellende Zeichenfolge
punkt: LOCATOR, Koordinatenwert

Beispiel:

TEXT ↑

Nach der Eingabe dieses Befehls wird der Benutzer durch den Promptstring TEXT: aufgefordert, den Textstring einzugeben, der später auf dem Bildschirm ausgegeben werden soll. Nach der Eingabe dieses Textes erscheint das Fadenkreuz. Nach dem Positionieren desselben ist der Anfangspunkt des Textstrings festzulegen und durch das Drücken irgendeiner alphanumerischen Taste zu quittieren. Danach erscheint der vorher eingegebene Textstring an dieser Stelle.

TEXT Teststring (1,3) ↑

C: TEXT Teststring (1,3)

Nach der Eingabe dieses Befehls wird der String : 'Teststring' bei der Koordinate x = 1 und y = 3 ausgegeben.

Bemerkung :

Die Farbe, der Font, die Zeichenhöhe, die Zeichenbreite und der Winkel unter dem der Text ausgegeben werden soll lassen sich durch den ATTRIBUTE Befehl vorher festlegen oder durch CHANGE nachträglich verändern. Will man im zweiten Beispiel einen Textstring mit Sonderzeichen (z.B. Blank) eingeben, so ist der String in Hochkommas (' ') einzuschließen.

AREA

Das AREA-Kommando erzeugt eine Füllfläche. Dabei werden die gerade gültigen AREA-Attribute verwendet.

Syntax:

AREA [punkt]*
punkt: LOCATOR, Randpunkt der Fläche.

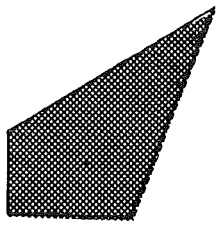
Beispiel:

AREA ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren desselben ist jedes Koordinatenpaar durch das Drücken irgendeiner alphanumerischen Taste einzugeben. Ab dem zweiten Koordinatenpunkt erscheint sofort die Verbindungsline zum letzen Punkt. Dieser Vorgang wird abgeschlossen, wenn eine der Tasten "E" oder "Q" gedrückt wird. Der letzte Punkt wird mit dem ersten verbunden, um die Umrandung der Fläche zu schließen. Die Fläche wird nun mit der aktuellen Farbe gefüllt.

AREA (7,5) (8,5) (10,10) (5,7) ↑

Nach der Eingabe dieses Befehls wird der geschlossener Polygonzug mit den Koordinaten :
Punkt 1 X = 7 , Y = 5
Punkt 2 X = 8 , Y = 5
Punkt 3 X = 10 , Y = 10
Punkt 4 X = 5 , Y = 7 und
Punkt 5 X = 7 , Y = 5 gezeichnet. Die Innenfläche wird dann mit der aktuellen Farbe eingefärbt.



C: AREA (5,5) (8,5) (10,10) (5,7)

Bemerkung :

Die Farbe der Fläche läßt sich durch den ATTRIBUTE Befehl vorher festlegen oder durch CHANGE nachträglich verändern.

CELL

Das CELL-Kommando erzeugt eine Zellmatrix.

Syntax:

CELL punkt1 punkt2 n m [farben]*

punkt1: LOCATOR, Koordinatenwert, linker unterer Eckpunkt der Zellmatrix

punkt2: LOCATOR, Koordinatenwert, rechter oberer Eckpunkt der Zellmatrix

n,m: INTEGER, Dimension der Zellmatrix

farbe: INTEGER, Zellmatrix. Werden weniger als n*m Farbwerte angegeben, werden die restlichen Werte mit der Hintergrundfarbe gefüllt.

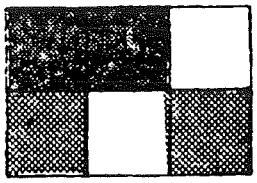
Beispiel:

CELL ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren desselben ist die linke untere und die rechte obere Ecke eines Rechteckes einzugeben. Das Quittieren erfolgt durch das Drücken irgendeiner alphanumerischen Taste. Danach wird die Umrandung des Rechteckes gezeichnet. Nun werden nacheinander die Dimensionen N und M der Zellmatrix durch die Promptstrings N: und M: angefordert. Nach erfolgter Eingabe erwartet nun das Programm N * M Farbwerte, die fortlaufend durch Blanks getrennt eingegeben werden müssen. Ist dies erfolgt, so erscheint nochmals die Eingabeaufforderung COLOUR: . Diese ist durch das Drücken der "RETURN"- oder "ENTER"-Taste abzubrechen. Jetzt wird die Zellmatrix in den angegebenen Farben gezeichnet.

CELL (1,1) (7,5) 3 2 65 91 105 85 205 212 ↑

Nach der Eingabe dieses Befehls wird ein Zellarray mit den Farben 65, 91, 105, 85, 205, 212 gezeichnet. Seine Eckpunkte liegen bei:
linker unterer Punkt X = 1 , Y = 1 und
rechter oberer Punkt X = 5 , Y = 5.
Die Eingabewerte 2 und 2 sind die Dimension N * M der Zellmatrix.



C: CELL (1,1) (7,5) 3 2 65 91 105 85 205 212

CIRCLE

Das CIRCLE-Kommando erzeugt einen Kreis, der entweder nur aus seinem Rand besteht oder gefüllt sein kann. Im ersteren Fall werden dabei die gerade gültigen POLYLINE-Attribute verwendet, für den gefüllten Kreis die gerade gültigen AREA-Attribute.

Syntax:

CIRCLE mpunkt radius [AREA]
mpunkt: LOCATOR, Mittelpunkt des Kreises
radius: REAL, Radius des Kreises
AREA bedeutet, daß der Kreis gefüllt werden soll. Der Kreis wird durch ein POLYLINE- oder ein AREA-Element angenähert, das Winkel-Inkrement dafür kann durch das ATTRIBUTE CIRCLE INCREMENT-Kommando geändert werden.

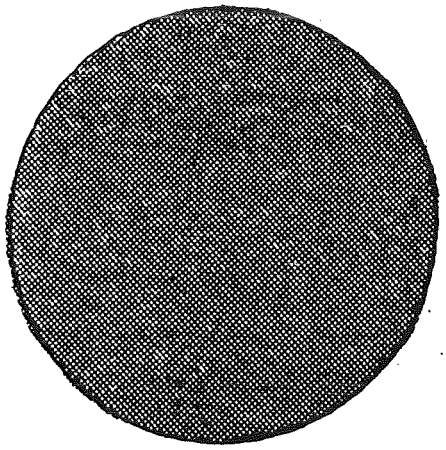
Beispiel:

CIRCLE ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren desselben ist eine alphanumerische Taste zu drücken. Dann erscheint der Prompt "RADIUS:", worauf über die alphanumerische Tastatur der Radius einzugeben ist, z.B. "1.0". Da die Angabe AREA wahlfrei ist, wird der Kreis nicht ausgefüllt erzeugt.

CIRCLE (10,10) 5.2 AREA ↑

Nach der Eingabe dieses Befehls wird ein gefüllter Kreis mit dem Mittelpunkt bei (10,10) und dem Radius 5.2 erzeugt.



C: CIRCLE (10,10) 5.2 AREA

ARC

Das ARC-Kommando erzeugt einen Kreisbogen, der entweder nur aus dem Bogen besteht oder gefüllt sein kann. Im ersteren Fall werden dabei die gerade gültigen POLYLINE-Attribute verwendet, im zweiten Fall wird der gefüllte Kreisbogen wird zu einem Kreissegment geschlossen, das mit den gerade gültigen AREA-Attributen dargestellt wird.

Syntax:

ARC mpunkt radius angle1 angle2 [AREA]
mpunkt: LOCATOR, Mittelpunkt des Kreises
radius: REAL, Radius des Kreises
angle1: REAL, Anfangswinkel des Kreises in Grad
angle2: REAL, Endwinkel des Kreises in Grad
AREA bedeutet, daß der Bogen gefüllt werden soll. Der Kreisbogen wird durch ein POLYLINE- oder ein AREA-Element angenähert, das Winkel-Inkrement dafür kann durch das ATTRIBUTE CIRCLE INCREMENT-Kommando geändert werden. Der Bogen wird vom Anfangswinkel in mathematisch positivem Sinne (also im Gegenuhrzeigersinn) zum Endwinkel erzeugt.

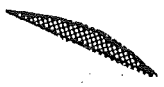
Beispiel:

ARC ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren desselben ist eine alphanumerische Taste zu drücken. Dann erscheint der Prompt "RADIUS:", worauf über die alphanumerische Tastatur der Radius einzugeben ist, z.B. "1.0". Der Anfangs- und Endwinkel werden danach mit dem Prompt "ANGLE:" angefordert. Da die Angabe AREA wahlfrei ist, wird der Bogen nicht ausgefüllt.

ARC 5.2 45 90 AREA ↑

Nach der Eingabe dieses Befehls wird der Mittelpunkt über den Steuerknüppel angefordert und dann ein gefülltes Kreissegment mit Radius 5.2 vom Winkel 45 Grad bis 90 Grad erzeugt.



C: ARC 5.2 45 90 AREA

SECTOR

Das SECTOR-Kommando erzeugt einen Kreissektor, der aus einem Kreisbogen und den beiden Verbindungslinien von den Bogenenden zum Mittelpunkt besteht. Der Kreissektor kann nur aus seiner Umrandung bestehen oder gefüllt sein. Im ersteren Fall werden dabei die gerade gültigen POLYLINE-Attribute verwendet, für den gefüllten Kreissektor die gerade gültigen AREA-Attribute.

Syntax:

SECTOR mpunkt radius angle1 angle2 [AREA]
mpunkt: LOCATOR, Mittelpunkt des Kreises
radius: REAL, Radius des Kreises
angle1: REAL, Anfangswinkel des Kreises in Grad
angle2: REAL, Endwinkel des Kreises in Grad
AREA bedeutet, daß der Sektor gefüllt werden soll. Der Kreissektor wird durch ein POLYLINE- oder ein AREA-Element angenähert, das Winkel-Inkrement dafür kann durch das ATTRIBUTE CIRCLE INCREMENT-Kommando geändert werden. Der Sektor wird vom Anfangswinkel in mathematisch positivem Sinne (also im Gegenuhrzeigersinn) zum Endwinkel erzeugt.

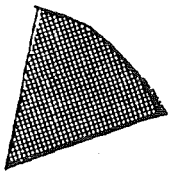
Beispiel:

SECTOR ↑

Nach der Eingabe dieses Befehls erscheint das Fadenkreuz. Nach dem Positionieren desselben ist eine alphanumerische Taste zu drücken. Dann erscheint der Prompt "RADIUS:", worauf über die alphanumerische Tastatur der Radius einzugeben ist, z.B. "1.0". Der Anfangs- und Endwinkel werden danach mit dem Prompt "ANGLE:" angefordert. Da die Angabe AREA wahlfrei ist, wird der Sektor nicht ausgefüllt.

SECTOR 3.6 AREA ↑

Nach der Eingabe dieses Befehls wird der Mittelpunkt über den Steuerknüppel angefordert. Der Radius ist 3.6, die Winkel werden über die Tastatur angefordert. Dann wird ein gefüllter Kreissektor mit Radius 3.6 erzeugt.



C: SECTOR 3.6 AREA

RECTANGLE
Das RECTANGLE-Kommando erzeugt ein Rechteck parallel zum Darstellungsfeld, das entweder nur aus seinem Rand besteht oder gefüllt sein kann. Im ersteren Fall werden dabei die gerade gültigen POLYLINE-Attribute verwendet, für das gefüllte Rechteck die gerade gültigen AREA-Attribute.

Syntax:
RECTANGLE punkt1 punkt2 [AREA]
punkt1: LOCATOR, linker unterer Eckpunkt des Rechteckes
punkt2: LOCATOR, rechter oberer Eckpunkt des Rechteckes
AREA bedeutet, daß das Rechteck gefüllt werden soll.
Das Rechteck wird durch ein POLYLINE- oder ein AREA-Element mit 5 bzw. 4 Punkten dargestellt.

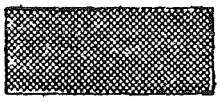
Beispiel:

RECTANGLE ↑

Nach der Eingabe dieses Befehls erscheint zweimal das Fadenkreuz. Dies ist jeweils mit dem Steuerknüppel zu positionieren, danach ist eine alphanumerische Taste zu drücken. Da die Angabe AREA wahlfrei ist, wird ein nicht ausgefülltes Rechteck erzeugt.

RECTANGLE (10,10) (15,12) AREA ↑

Nach der Eingabe dieses Befehls wird ein Füllgebiet mit den Randpunkten (10,10), (15,10), (15,12) und (10,12) erzeugt.



C: RECTANGLE (10,10) (15,12) AREA

2.2.4 Kommandos für Attribute

Der Editor E82 verwaltet für die Grundelemente POLYLINE, POLYMARKER, TEXT und AREA jeweils Mengen von Attributen. Diese Attribute werden durch das ATTRIBUTE-Kommando gesetzt. Die Realisierung der Attribute hängt natürlich von den Fähigkeiten des graphischen Arbeitsplatzes ab. Während GKS zwischen arbeitsplatzabhängigen und unabhängigen Attributen unterscheidet, benutzt der Editor E82 nur einen Satz von Attributen pro Grundelement. Der Benutzer wird informiert, wenn die vorhandenen Fähigkeiten nicht ausreichen, um die gewünschten Attribute zu verwirklichen. Die Farbe wird durch einen Farbindex angesprochen. Jedem solchen Farbindex ist im GKS und im E82 über eine Farbtabelle eine ganz bestimmte Farbe zugeordnet. Die Farbtabelle wird zu Beginn des Arbeitens mit dem E82 mit Farbwerten vorbelegt. Es ist aber auch möglich, durch das COLOUR-Kommando die Farbtabelle zu verändern, obwohl das nur in speziellen Fällen sinnvoll sein dürfte.

Die Abbildungen 3 bis 6 zeigen Beispiele für die Attribute für die Darstellungselemente POLYLINE, POLYMARKER, TEXT, AREA und CIRCLE. Die Elemente ARC, CIRCLE, RECTANGLE und SECTOR benutzen die POLYLINE-Attribute, wenn sie als Vektorelemente erzeugt werden, und die AREA-Attribute, wenn sie gefüllt sind. Tabelle 1 gibt einen Überblick über die verfügbaren Zeichensätze.

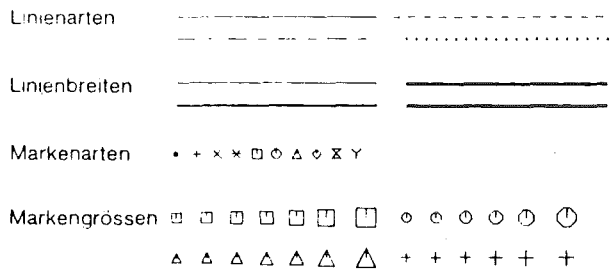


Abbildung 3: Attribute für POLYLINE und POLYMARKER

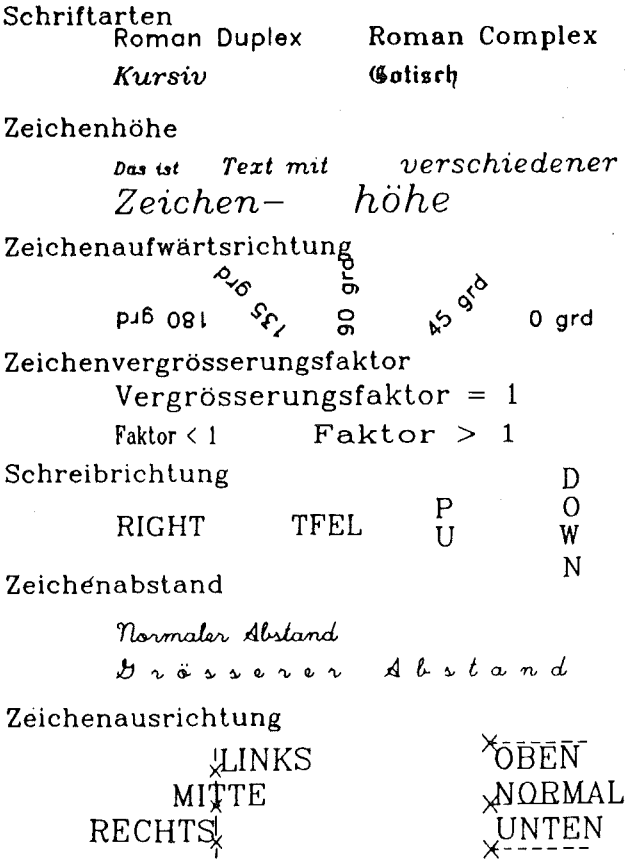


Abbildung 4: Attribute für TEXT

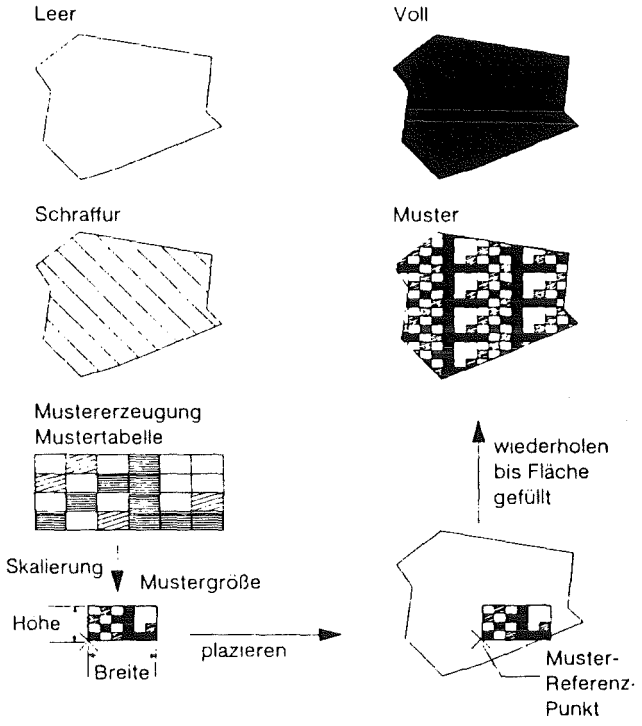


Abbildung 5: Attribute für AREA

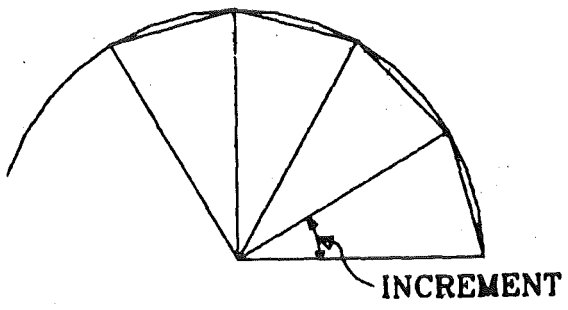


Abbildung 6: Inkrement für CIRCLE, ARC und SECTOR

FONT	PRECISION	Zeichensatzname	Beispiel für den Zeichensatz
1	STROKE	Standard Software	
2	STRING	Standard Hardware	
3	STROKE	AED2 (gefüllt)	
4	STROKE	Software griechisch	
5	STROKE	AED1 (nur Rand)	
6	STROKE	AED3 (schraffiert)	
7	STROKE	Hershey 1, simplex roman	simplex ROMAN
8	STROKE	Hershey 2, duplex roman	DUPLEX roman
9	STROKE	Hershey 3, complex roman	Complex 1234
10	STROKE	Hershey 4, triplex roman	triplex ZYXW
11	STROKE	Hershey 5, complex italic	*Kursivschrift*
12	STROKE	Hershey 6, triplex italic	*Triplex KURSIV*
13	STROKE	Hershey 7, simplex script	Schreibschrift
14	STROKE	Hershey 8, complex script	Complex Script
15	STROKE	Hershey 9, simplex greek	Σιμπλεξ Γρεεκ
16	STROKE	Hershey 10, complex greek	Γριεχηισχη Κωμπλεξ
17	STROKE	Hershey 11, old English	Old English
18	STROKE	Hershey 12, old German	Deutsch Gotisch
19	STROKE	Hershey 13, old Italian	Italiano viaggio
20	STROKE	Hershey 14, complex cyrillic	Русскии Москва

Tabelle 1: Verfügbare Zeichensätze

Anmerkung: Die verfügbaren Zeichensätze und ihre Zuordnung zu den Zeichensatz-Nummern ist zur Zeit der Drucklegung dieses Berichtes noch nicht endgültig festgelegt. Die obige Tabelle hat daher vorläufigen Charakter.

ATTRIBUTE

Das ATTRIBUTE-Kommando dient der Änderung der aktuellen Attribute für eine Grundelement-Art. Die Attribute gelten dann für alle danach erzeugten Grundelemente dieser Art.

Syntax:

```
ATTRIBUTE
  { POLYLINE [ polyline_attribute ]* |
    POLYMARKER [ polymarker_attribute ]* |
    TEXT [ text_attribute ]* |
    AREA [ area_attribute ]* |
    CIRCLE circle_attribute }

polyline_attribute ::= { TYPE ltype | WIDTH lwidth | COLOUR index }
polymarker_attribute ::= { HEIGHT mheight | TYPE mtype | COLOUR index }
text_attribute ::= { HEIGHT theight | RATIO ratio | SPACING spacing |
   ANGLE angle | COLOUR index | PRECISION prec | FONT font }
area_attribute ::= { STYLE style | WINDOW punkt1 punkt2 |
   PATTERN n m [farbe]* | COLOUR index }
circle_attribute ::= INCREMENT cangle
```

ltype: INTEGER, Linienart (Standard: 1)
lwidth: REAL, Strichstärke(Standard: 1)
index: INTEGER, Farbindex
mheight: REAL, Markenhöhe
mtype: INTEGER, Markentyp
theight: REAL, Zeichenhöhe
ratio: REAL, Breite-zu-Höhe-Verhältnis (Standard: 1)
spacing: REAL, Zeichenzwischenraum (Standard: 0)
angle: REAL, Winkel der Textgrundlinie in Grad (Standard: 0)
prec ::= { CHAR | STRING | STROKE }, Textqualität
font: INTEGER, Zeichensatz (Standard: 1)
style: { HOLLOW | PATTERN | FILLED | HATCH }, Füllart
punkt1: LOCATOR, Koordinatenwert, linker unterer Eckpunkt der Mustermatrix
punkt2: LOCATOR, Koordinatenwert, rechter oberer Eckpunkt der Mustermatrix
cangle: REAL, Winkelinkrement zum Umwandeln eines Kreises in einen Polygonzug (in Grad)
n,m: INTEGER, Dimension der Mustermatrix
farbe: INTEGER, Fardindices der Mustermatrix. Werden weniger als n x m Indices angegeben, werden die restlichen Werte der Matrix nicht verändert.

Beispiel:

ATTRIBUTE POLYLINE COLOUR 206 ↑

Nach der Eingabe dieses Befehls wird die Farbe für die Polylines auf den Wert 206 geändert. Alle nachfolgend gezeichneten Polygonzüge werden dann mit dieser Farbe gezeichnet.

ATTRIBUTE TEXT FONT 3 ↑

Nach der Eingabe dieses Befehls wird in Zukunft bei der Textausgabe der Font 3 verwendet. Dieser enthält Softwarecharacters. Er ermöglicht Texte mit einer höheren Qualität zu erzeugen.

ATTRIBUTE TEXT HEIGHT 0.8 RATIO 1.2 ↑

Nach der Eingabe dieses Befehls wird in Zukunft bei der Textausgabe die Texthöhe auf 0.8 und die Breite auf das 1.2-fache der zu der Texthöhe 0.8 gehörigen Normalbreite gesetzt.

BACK
Das BACK-Kommando ändert die Hintergrundfarbe. Es wird erst wirksam, wenn das Bild neu gezeichnet wird.

Syntax:
BACK colour
colour: INTEGER, Farbindex, in den die Hintergrundfarbe geändert werden soll

Beispiel:

BACK 14 ↑

Nach der Eingabe dieses Befehls wird die Farbe 14 als Hintergrundfarbe abgespeichert. Erst wenn das Bild neu gezeichnet wird, oder bei der Anwendung des CLEAR Befehls wird diese Farbe verwendet.

COLOUR
Das COLOUR-Kommando dient der Änderung von Einträgen in der Farbtabelle. Das bedeutet, daß alle Grundelemente, die den jeweiligen Farbindex besitzen, ihre Farbe entsprechend ändern. Die Werte können von einer Datei gelesen werden, auf denen sie gespeichert sind. Das Format ist im Abschnitt 6.3 angegeben.

Syntax:

COLOUR { STANDARD | FILE fname | index [red green blue]* }
STANDARD: Die Standard-Farbtabelle des E82 wird restauriert.
FILE: Die komplette Farbtabelle wird von einer Datei eingelesen.
fname: STRING(8), Datei mit der Farbtabelle.
index: INTEGER, Farbindex, ab dem die Farbtabelle gefüllt bzw. geändert werden soll.
red: REAL, Rotwert der Farbe 0 <= red <= 1
green: REAL, Grünwert der Farbe 0 <= green<= 1
blue: REAL, Blauwert der Farbe 0 <= blue <= 1

Beispiel:

COLOUR 206 127 127 127 ↑

Nach der Eingabe dieses Befehls wird die Farbe die in der Farbtabelle im Index 206 abgespeichert ist mit der Rot-, Grün- und Blauwerten von 127 überschrieben. Dies hat zur Folge, dass alle mit dieser Farbe (206) gezeichneten Bildteile die neue Farbe annehmen.

CHANGE
Das CHANGE-Kommando dient der Änderung der Attribute eines bereits existierenden Grundelementes oder einer Gruppe von Grundelementen. Die Grundelemente werden durch ihren Segmentnamen identifiziert, d.h. normalerweise durch die PICKER-Eingabe angepickt. Attributangaben gelten nur für diejenigen Grundelemente, die dieses Attribut besitzen.

Syntax:

CHANGE [segment]*
{ POLYLINE [polyline_attribute]* |
POLYMARKER [polymarker_attribute]* |
TEXT [text_attribute]* |

AREA [area_attribute]* |
CIRCLE circle_attribute }

segment: SEGMENT, Segment, dessen Attribute geändert werden sollen.
polyline_attribute ::= { TYPE ltype | WIDTH lwidth | COLOUR index }
polymarker_attribute ::= { HEIGHT mheight | TYPE mtype | COLOUR index }
text_attribute ::= { HEIGHT theight | RATIO ratio | SPACING spacing |
ANGLE angle | COLOUR index | PRECISION prec | FONT font }
area_attribute ::= { STYLE style | WINDOW punkt1 punkt2 |
PATTERN n m [farbe]* | COLOUR index }
circle_attribute ::= INCREMENT cangle

ltype: INTEGER, Linienart (Standard: 1)
lwidth: REAL, Strichstärke(Standard: 1)
index: INTEGER, Farbindex
mheight: REAL, Markenhöhe
mtype: INTEGER, Markentyp
theight: REAL, Zeichenhöhe
ratio: REAL, Breite-zu-Höhe-Verhältnis (Standard: 1)
spacing: REAL, Zeichenzwischenraum (Standard: 0)
angle: REAL, Winkel der Textgrundlinie in Grad (Standard: 0)
prec ::= { CHAR | STRING | STROKE }, Textqualität
font: INTEGER, Zeichensatz (Standard: 1)
style: { HOLLOW | PATTERN | FILLED | HATCH }, Füllart
punkt1: LOCATOR, Koordinatenwert, linker unterer Eckpunkt der Mustermatrix
punkt2: LOCATOR, Koordinatenwert, rechter oberer Eckpunkt der Mustermatrix
cangle: REAL, Winkelinkrement zum Umwandeln eines Kreises in einen Polygonzug (in Grad)
n,m: INTEGER, Dimension der Mustermatrix
farbe: INTEGER, Fardindices der Mustermatrix. Werden weniger als n x m Indices angegeben, werden die restlichen Werte der Matrix nicht verändert.

Beispiel:

CHANGE POLYLINE COLOUR 199 ↑

Nach der Eingabe dieses Befehls erscheint das Positionierungskreuz, mit dem ein Segment oder mehrere Segmente gepickt werden können. Alle in diesen Segmenten enthaltenen Polylines erhalten die neue Farbe 199.

CHANGE TEXT FONT 3 HEIGHT 0.8 RATIO 1.2 ↑

Nach der Eingabe dieses Befehls können ebenfalls Segmente gepickt werden. In ihnen enthaltene Texte erhalten den Zeichensatz 3, die Höhe 0.8 und das Breite-zu-Höhe-Verhältnis 1.2.

2.2.5 Kommandos für Transformationen

Jedes Segment kann gedreht, skaliert oder verschoben werden. Nach Erzeugung eines Grundelementes ist dieses gleichzeitig das aktuelle Segment. Das aktuelle Segment kann über den Namen S%0 angesprochen werden. Alle Transformationen können inkremental oder absolut sein. Die inkrementalen Transformationen beziehen sich auf den aktuellen Zustand, die absoluten auf den Urzustand. Die Wahl der gewünschten Möglichkeit erfolgt mit dem TMODE-Kommando.

TRANSLATE
Das TRANSLATE-Kommando dient der Verschiebung von Segmenten.

Syntax:

TRANSLATE [segment]* punkt
segment: SEGMENT, Segmente, die verschoben werden sollen.
punkt: LOCATOR, Verschiebevektor

Beispiel:

TRANSLATE ↑

Nach der Eingabe dieses Befehls erscheint das Positionierungskreuz, mit dem ein Segment oder mehrere Segmente gepickt werden können. Der Pickvorgang wird durch das Drücken der Q- oder E-Taste abgeschlossen. Dann erscheint das Fadenkreuz für den Verschiebevektor. Nach Drücken einer Taste werden alle gepickten Segmente um den Verschiebevektor verschoben.

TEXT 'xyz' (12,5) ↑
TRANSLATE S%0 (10,10) ↑

Mit dem TEXT-Kommando wird ein Text bei (12,5) erzeugt und zum aktuellen Segment gemacht. Mit dem zweiten Kommando wird der Text um (10,10) verschoben, so daß er bei (22,15) liegt.

ROTATE
Das ROTATE-Kommando dient der Drehung von Segmenten um den Mittelpunkt des aktuellen Windows.

Syntax:

ROTATE [segment]* grad
segment: SEGMENT, Segmente, die verschoben werden sollen.
grad: REAL, Drehwinkel in Grad

ROTP
Das ROTP-Kommando dient der Drehung von Segmenten um einen Punkt.

Syntax:

ROTP [segment]* grad punkt
segment: SEGMENT, Segmente, die verschoben werden sollen.
grad: REAL, Drehwinkel in Grad
punkt: LOCATOR, Koordinatenwert des Drehpunktes

SCALE
Das SCALE-Kommando dient der Skalierung von Segmenten bezüglich des Mittelpunktes des aktuellen Windows.

Syntax:

SCALE [segment]* sx [sy]
segment: SEGMENT, Segmente, die verschoben werden sollen.
sx,sy REAL, Skalierungsfaktoren (falls sy fehlt, ist sy=sx)

SCALEP
Das SCALEP-Kommando dient der Skalierung von Segmenten bezüglich eines Punktes.

Syntax:

SCALE segment [segment]* sx sy punkt
segment: SEGMENT, Segmente, die verschoben werden sollen.
sx,sy REAL, Skalierungsfaktoren
punkt: LOCATOR, Koordinatenwert des Fixpunktes

WINDOW
Das WINDOW-Kommando legt den Anwenderkoordinatenbereich fest. Dies ist wichtig, wenn Koordinaten durch Werte angegeben werden sollen. Das Window wird standardmäßig auf den gesamten Viewport abgebildet. Wurde mit dem VIEWPORT-Kommando nur ein Teilbereich der gesamten Darstellungsfläche ausgewählt, wird das Window auf diesen Teilbereich, den Viewport, abgebildet.

Syntax:
WINDOW { EXTENT | punkt1 punkt2 }
EXTENT: Das Window wird gleich der aktuellen Bildausdehnung gesetzt.
punkt1: LOCATOR, Koordinatenwert, linker unterer Eckpunkt des Window
punkt2: LOCATOR, Koordinatenwert, rechter oberer Eckpunkt des Window

VIEWPORT
Das VIEWPORT-Kommando dient zur Wahl eines kleineren als des maximal möglichen Bildbereiches zur Darstellung des Bildes.

Syntax:
VIEWPORT { MAX | punkt1 punkt2 }
MAX: Der Viewport wird auf den maximal möglichen Wert gesetzt.
punkt1: LOCATOR, Koordinatenwert, linker unterer Eckpunkt des Viewports
punkt2: LOCATOR, Koordinatenwert, rechter oberer Eckpunkt des Viewports.

ASPECT
Das ASPECT-Kommando dient dazu, bei der Window/Viewport-Transformation die Einhaltung eines einheitlichen Höhe/Breite-Verhältnisses zu verlangen oder nicht. Bei ASPECT ON wird das Window so verkleinert, daß es bei einheitlicher Skalierung zentriert in den Viewport hineinpaßt. Abbildung 7 gibt ein Beispiel dafür.

Syntax:
ASPECT {ON | OFF}
ON: Verzerrung ist nicht erlaubt
OFF: Verzerrung ist erlaubt.

Beispiel:
```
WINDOW (10,10) (30,20) ↑
VIEWPORT MAX ↑
ASPECT ON ↑
```

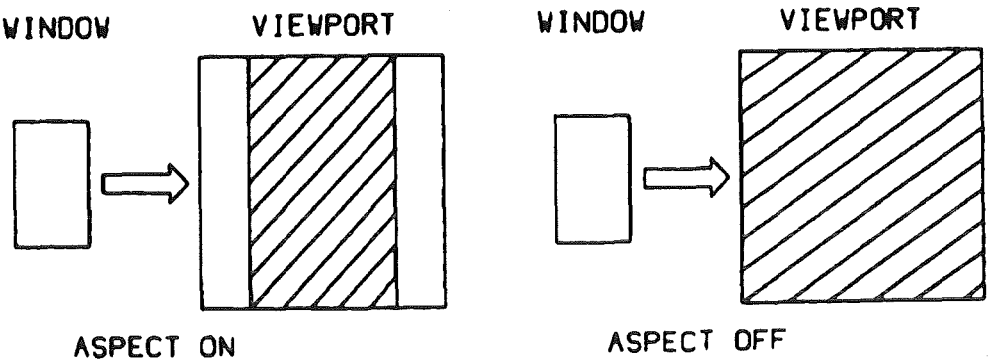


Abbildung 7: Window/Viewport-Transformation mit ASPECT ON oder OFF

ZOOM
Das ZOOM-Kommando dient zum Vergrößern der Darstellung des gesamten Bildes auf der Darstellungsfläche. Das interne Bild wird dadurch nicht verändert. Jedes ZOOM kann durch ZOOM 1 wieder rückgängig gemacht werden. Das ZOOM-Kommando bewirkt ein Software-Zoom mit Hilfe des GKS. Während sich WINDOW und VIEWPORT nur auf zukünftig erzeugte Darstellungselemente auswirken, wirken ZOOM und PAN auf das aktuelle Bild. Das Bild wird dazu gelöscht und neu generiert. Abbildung 8 zeigt ein Beispiel für ZOOM und PAN.

Syntax:
```
ZOOM zfact
```
zfact: REAL, Zoomfaktor, >1 bewirkt Vergrößerung, <1 Verkleinerung des Bildes, ausgehend von der durch ZOOM 1 definierten Größe.

Beispiel:

```
ZOOM 2.0 ↑
```

PAN
Das PAN-Kommando dient zum Verschieben der Darstellung des gesamten Bildes auf der Darstellungsfläche. Das ist vor allem bei Zoomfaktoren >1 sinnvoll, da dann der sichtbare vergrößerte Bildausschnitt über den Bildschirm verschoben werden kann. Das interne Bild wird dadurch nicht verändert. Jedes PAN kann durch ZOOM 1 oder PAN (0,0) wieder rückgängig gemacht werden. Das PAN-Kommando bewirkt ein Software-Pan mit Hilfe des GKS.

Syntax:
```
PAN punkt
```
punkt: LOCATOR, neuer Nullpunkt. Der angegebene Punkt wird auf den Mittelpunkt des Bildschirmes gelegt.

Beispiel:

PAN ↑

Mit Hilfe des Positionierungskreuzes kann ein Punkt angefahren werden, der dann zum Mittelpunkt des Bildschirmes verschoben wird.

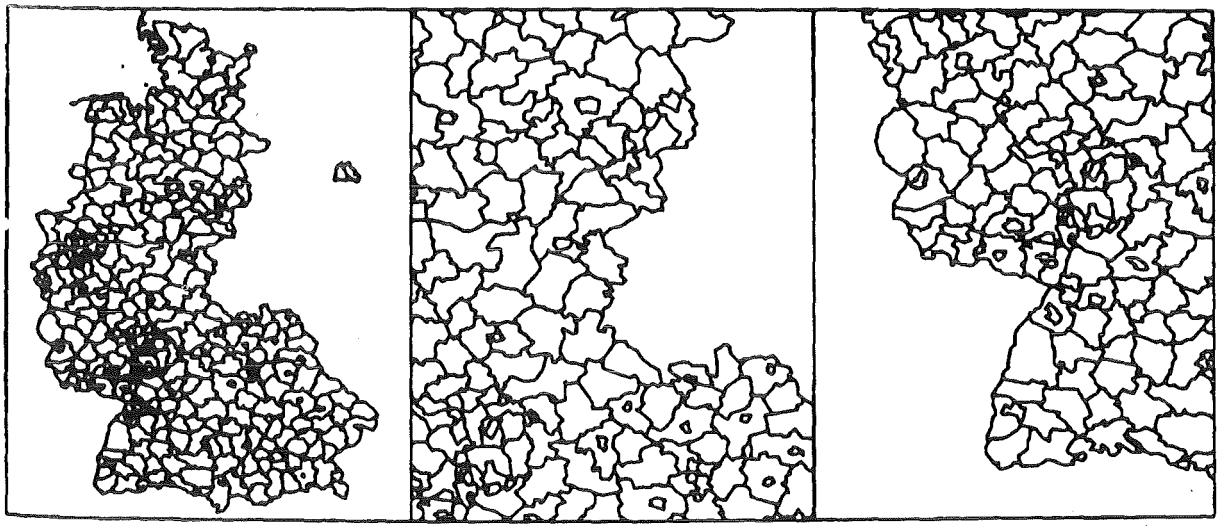


Abbildung 8: Beispiel für ZOOM und PAN

TMODE

Das TMODE-Kommando dient zum Umschalten des Transformationsmodus der Operationen TRANSLATE, ROTATE, ROTP, SCALE und SCALEP auf inkremental oder absolut. Im inkrementalen Modus wird jede Transformation auf die aktuelle Lage angewandt. Eine Verschiebung um (0,0) bewirkt daher z.B. in diesem Modus keine Veränderung. Im absoluten Modus wird jede Transformation auf die ursprüngliche Lage eines Darstellungselementes angewandt. Eine Verschiebung um (0,0) bewirkt daher die Positionierung des Elementes an dem Punkt, wo es ursprünglich erzeugt worden war.

Syntax:

```
TMODE { INC | ABS }
INC:      Transformationsmodus inkremental
ABS:      Transformationsmodus absolut
```

GRID

Das GRID-Kommando dient zur Angabe eines Rasters für alle LOCATOR-Eingaben in folgenden Kommandos. Alle Eingaben werden auf den nächsten Rasterwert auf- oder abgerundet. Damit wird es einfacher, waagrechte oder senkrechte Linien zu ziehen oder äquidistante Koordinaten anzufahren.

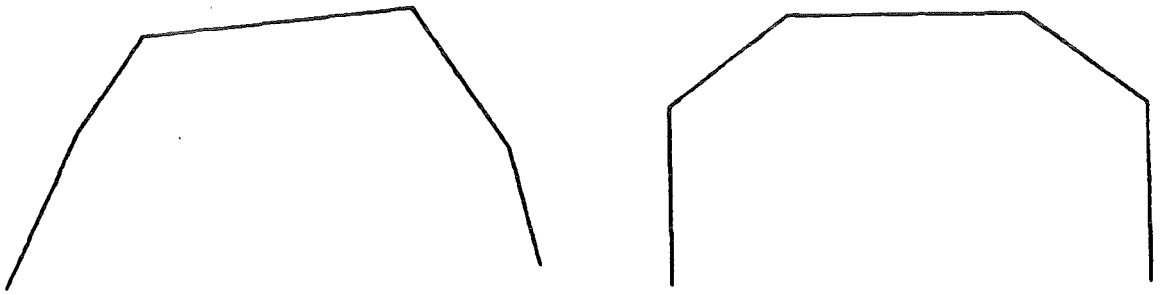
Syntax:

```
GRID { xgrid [ygrid] | OFF }
xgrid:    REAL, Rasterinkrement in x-Richtung
ygrid:    REAL, Rasterinkrement in y-eichtung
OFF:      Die Rasterung wird abgeschaltet.
```

Beispiel:

GRID 2 1.5 ↑
POLYLINE (1.1,1.5) (2.3,4.2) (3.4,5.8) (7.9,6.3) (9.5,4) (10,2) ↑

Durch das GRID-Kommando werden alle x-Werte auf Vielfache von 2 und alle y-Werte auf Vielfache von 1.5 gerundet. Der Polygonzug erhält dadurch die Koordinaten: (2,1.5) (2,4.5) (4,6) (8,6) (10,4.5) (10,1.5). Abbildung 9 zeigt den Polygonzug mit und ohne GRID.



C:POLYLINE (1.1,1.5) (2.3,4.2) (3.4,5.8) (7.9,6.3) (9.5,4) (10,2)

Abbildung 9: Polygonzug mit und ohne Raster

2.2.6 Kommandos für Bilder und Segmente

Die folgenden Kommandos dienen zum Lesen, Schreiben oder Löschen von Bildern, zum Zusammenfassen von Darstellungselementen zu Segmenten und zum Auftrennen eines Segmentes in einzelne Darstellungselemente.

CLEAR
Das CLEAR-Kommando löscht das aktuelle Bild.

Syntax:
CLEAR

Beispiel:

CLEAR ↑

Ist ein aktuelles Bild vorhanden, das noch nicht auf eine Bilddatei gerettet wurde, so erscheint die Aufforderung: "CONFIRM DELETE PICTURE". Danach ist noch einmal die RETURN-Taste zu drücken. Jede andere Eingabe hebt den CLEAR-Befehl auf.

RESHOW
Das RESHOW-Kommando löscht den Bildschirm und zeichnet das aktuelle Bild neu.

Syntax:
RESHOW

Beispiel:

RESHOW ↑

SAVE

Das SAVE-Kommando rettet das aktuelle Bild auf die aktuelle Bilddatei. Dies ist vor allem eine Maßnahme gegen Systemzusammenbrüche. Das gerettete Bild überschreibt ein dort schon vorhandenes Bild.

Syntax:
SAVE

RESTORE

Das RESTORE-Kommando restauriert das aktuelle Bild von der aktuellen Bilddatei. Falls das Bild seit dem letzten SAVE verändert wurde, muß zuvor ein CLEAR erolgen.

Syntax:
RESTORE

Beispiel:

RESTORE ↑

Falls noch ein aktuelles Bild existiert, das noch nicht auf eine Bilddatei gerettet wurde, erscheint die Aufforderung "CLEAR OR SAVE CURRENT PICTURE", sonst wird das Bild von der aktuellen Bilddatei geholt.

COPY

Das COPY-Kommando kopiert ein Segment und verschiebt es an eine gewünschte Stelle.

Syntax:
COPY segment position
segment: SEGMENT, zu kopierendes Segment
position: LOCATOR, Position des zu kopierenden Segmentes.

Beispiel:

COPY ↑

Das Positionierungskreuz erscheint einmal zum Picken des Segmentes und einmal zum Positionieren der Kopie des Segmentes.

DELETE

Das DELETE-Kommando löscht eine Reihe von Segmenten.

Syntax:

DELETE [segment]*
segment: SEGMENT, zu löschendes Segment

USE

Das USE-Kommando dient zur Angabe von Bilddateien zum Einlesen und zur Ausgabe von Bildern. Die Dateien werden über ihren Filenamen identifiziert. Falls nicht die Standard-Files benutzt werden, müssen sie vorher allokiert werden (siehe Abschnitt 2.4.10). Das USE hat eine wichtige Wirkung auf die Datei: sie wird eröffnet. Wenn sie schon offen ist, wird sie geschlossen und neu eröffnet. Für Ausgabe-Bilddateien heißt das, daß die Datei wieder von vorne geschrieben wird und alle auf ihr befindlichen Bilder verloren sind. Falls der Editor E82 feststellt, daß dies geschieht, wird der Benutzer vorher gewarnt. Für Eingabe-Bilddateien hat das USE (das dadurch bewirkte Öffnen der Datei) zur Folge, daß der Zeiger in der Datei vor dem ersten Bild steht. Das hat Auswirkungen auf das READ (siehe unten). Ein USE wird durch ein neues USE rückgängig gemacht.

Syntax:

```
USE { INPUT [ fname ]* |
      OUTPUT fname }
```

INPUT: Die angegebenen Files werden als Eingabedatei für nachfolgende READ- oder INSERT-Kommandos benutzt.

OUTPUT: Der angegebene File wird als Ausgabedatei für nachfolgende WRITE-Kommandos benutzt.

fname: CHARACTER(8), Filename

Beispiel:

```
%ALLOC FIN DSN 'TSO147.E82IN.DATA' ↑
%ALLOC FIN1 DSN 'IRE913.DIA.DATA(M1)' ↑
%ALLOC FOUT DSN 'TSO959.NEW.DATA' ↑
USE INPUT FIN FIN1 ↑
USE OUTPUT FOUT ↑
```

Das READ-Kommando wird danach die Bilder von den Dateien mit den E82-File-Namen FIN und FIN1, also von den Datasets mit den DS-Namen TSO147.E82IN.DATA und IRE913.DIA.DATA lesen. Das WRITE-Kommando wird Bilder auf die Datei mit dem Filenamen FOUT schreiben, also auf den Dataset mit dem DS-Namen TSO959.NEW.DATA.

WRITE

Das WRITE-Kommando schreibt ein Bild auf die Ausgabe-Bilddatei.

Syntax:

```
WRITE bname
```

bname: CHARACTER(8), Bildname

Beispiel:

```
WRITE 'Bild10' ↑
```

READ

Das READ-Kommando liest ein Bild von der Eingabe-Bilddatei. Das Bild wird zum aktuellen Bild. Sind mehrere Files als Eingabedatei definiert (durch USE INPUT), werden sie in der angegebenen Reihenfolge verkettet und wie eine einzige Bilddatei behandelt.

Syntax:
READ bname [{ NO number | FIRST | NEXT | LAST }]
bname: CHARACTER(8), Bildname. Der Bildname '#' ist ein allgemeiner Name, der auf jedes Bild zutrifft.
number: Bildnummer, falls der Bildname mehr als einmal auf der Bilddatei vorkommt, wird das number-te Bild dieses Namens eingelesen.
FIRST: Das erste Bild des Namens, das auf der Eingabedatei ist, wird eingelesen.
NEXT: Das nächste Bild des Namens wird eingelesen.
LAST: Das letzte Bild des Namens, das auf der Eingabedatei ist, wird eingelesen.
Standardwert ist NEXT.

Beispiel:

READ 'Bild10' ↑
READ '#' ↑
READ '#' FIRST ↑

Im ersten Fall wird das nächste Bild mit dem Namen 'Bild10' von der Eingabebildatei gelesen, im zweiten Fall das nächste Bild und im dritten Fall das erste Bild der Datei.

INSERT

Das INSERT-Kommando liest ein Bild von der Eingabe-Bilddatei und fügt es in das aktuelle Bild ein. Das gesamte eingelesene Bild wird im aktuellen Bild zu einem Segment, speziell auch zum aktuellen Segment.

Syntax:
INSERT { ACT | bname [{ NO number | FIRST | NEXT | LAST }]}
bname: CHARACTER(8), Bildname.
ACT: Füge das Bild von der aktuellen Bilddatei ein.
number: Bildnummer, falls der Bildname mehr als einmal auf der Bilddatei vorkommt, wird das number-te Bild dieses Namens eingelesen.
FIRST: Das erste Bild des Namens, das auf der Eingabedatei ist, wird eingelesen.
NEXT: Das nächste Bild des Namens wird eingelesen.
LAST: Das letzte Bild des Namens, das auf der Eingabedatei ist, wird eingelesen.
Standardwert ist NEXT.

COMPRESS

Das COMPRESS-Kommando komprimiert eine Bilddatei, indem von jedem Bildname nur noch ein einziges Exemplar behalten wird. Dabei wird jeweils das letzte Exemplar behalten. Eine offene Ausgabe-Datei (USE OUTPUT) kann nicht komprimiert werden.

Syntax:
COMPRESS fname
fname: CHARACTER(8), Filename

SEGMENT
Das SEGMENT-Kommando dient zum Zusammenfassen bestehender Segmente zu einem einzigen Segment.

Syntax:

SEGMENT [segment]*
   segment: SEGMENT, zusammenzufassende Segmente.

PRIMITIVE
Das PRIMITIVE-Kommando dient zum Aufteilen eines Segmentes in einzelne Segmente, die jeweils nur ein einziges Grundelement enthalten.

Syntax:
PRIMITIVE segment
   segment: SEGMENT, aufzuteilendes Segment.

2.2.7 Workspace-Kommandos

WORKSPACE
Das WORKSPACE-Kommando dient zum Einlesen oder Abspeichern eines Editor-Workspaces.

Syntax:
WORKSPACE { READ | WRITE } fname
   fname: CHARACTER(8), Filename des Workspaces.

Beispiel:

```
%ALLOCATE WS3 DSN 'TSO745.MYWORK.DATA(WS3)'
WORKSPACE READ WS3 ↑
```

Die Workspace-Werte für das Editieren von Bildern werden von der Datei WS3 (also vom Dataset mit dem Namen TSO745.MYWORK.DATA(WS3)) in die entsprechenden Datenbereiche des Editors E82 gelesen. Sie werden von nun an benutzt.

INQUIRE
Das INQUIRE-Kommando dient zum Abfragen von Werten des Workspaces des E82-Bildeditors. Die Werte werden auf den Bildschirm ausgegeben.

Syntax:
INQUIRE

```
{ POLYLINE [ polyline_attribute ]* |
  POLYMARKER [ polymarker_attribute ]* |
  TEXT [ text_attribute ]* |
  AREA [ area_attribute ]* |
  CIRCLE circle_attribute |
  BACK |
  COLOUR index number |
  WINDOW |
  VIEWPORT |
```

```
      ASPECT |
      ZOOM |
      PAN |
      TMODE |
      GRID |
      INPUT |
      OUTPUT      }

   polyline_attribute ::= { TYPE | WIDTH | COLOUR }
   polymarker_attribute ::= { HEIGHT | TYPE | COLOUR }
   text_attribute ::= { HEIGHT | RATIO | SPACING |
      ANGLE | COLOUR | PRECISION | FONT }
   area_attribute ::= { STYLE | WINDOW | PATTERN | COLOUR }
   circle_attribute ::= INCREMENT
   index: Farbindex, von dem an die Rot-, Grün- und Blauwerte aus der
      Farbtabelle gelistet werden sollen.
   number: Anzahl der abgefragten Farbindices.
   Zur Bedeutung der einzelnen Parameter siehe die betreffenden
   Kommandos zum Setzen der Werte, vor allem ATTRIBUTE, WINDOW, VIEWPORT,
   ASPECT, GRID, TMODE, BACK, COLOUR, ZOOM, PAN und USE.

Beispiel:

   ATTRIBUTE AREA STYLE FILLED ↑
   INQUIRE AREA STYLE ↑
   Ergebnis: Auf dem Bildschirm erscheint "FILLED".

   BACK 222 ↑
   INQUIRE BACK ↑
   Ergebnis: Auf dem Bildschirm erscheint "222".

   COLOUR 10 0 0 100 0 100 0 100 0 0 ↑
   INQUIRE COLOUR 10 3 ↑
   Ergebnis: Auf dem Bildschirm erscheint "10:(0 0 100) 11:(0 100 0)
   12:(100 0 0)".
```

2.2.8 Kommandos für AGF-Plotfiles

Es gibt zwei Kommandos zum Wandeln von AGF-Plotfiles in GKS-Metafiles und umgekehrt. Das soll die Integration bestehender Bilddateien in den Editor E82 erleichtern.

AGFGKS

Das AGFGKS-Kommando dient zum Wandeln von AGF-Plotfiles oder einzelner Bilder daraus in GKS-Bilddateien. Diese können dann mit USE, READ, INSERT weiter behandelt werden.

Syntax:

```
   AGFGKS agffile [nr] gksfile
   agffile:  CHARACTER(8), Filename des AGF-Plotfiles,
   gksfile:  CHARACTER(8), Filename der GKS-Bilddatei.
   nr:       INTEGER, Bildnummer des zu wandelnden Bildes. Falls sie
      fehlt, wird die gesamte Bilddatei gewandelt.
```

Beispiel:
```
%ALLOC F1 DSN 'TSO147.AGF.DATA' ↑
%ALLOC F2 DSN 'TSO147.GKS.DATA' ↑
AGFGKS F1 F2 ↑
```

GKSAGF

Das GKSAGF-Kommando dient zum Wandeln von GKS-Bilddateien oder einzelner Bilder daraus in AGF-Plotfiles. Diese können dann mit den vorhandenen Plotfile-Programmen weiter behandelt werden.

Syntax:
```
GKSAGF gksfile [nr] agffile
gksfile:  CHARACTER(8), Filename der GKS-Bilddatei,
agffile:  CHARACTER(8), Filename des AGF-Plotfiles.
nr:       INTEGER, Bildnummer des zu wandelnden Bildes. Falls sie
   fehlt, wird die gesamte Bilddatei gewandelt.
```

Beispiel:
```
AGFGKS F2 12 F1 ↑
```

2.3 Präsentationsgraphik

2.3.1 Daten und ihre Darstellung

Der Datenrepräsentationsmodul E82/VIEW ist ein Teilmodul des graphischen Editors E82. Er dient der graphischen Darstellung von Werten, die aus Rechen- oder Meßergebnissen gewonnen wurden. Die Werte sind auf Dateien gespeichert, die normalerweise in einem Standardformat vorliegen, das der Präsentationsmodul verarbeiten kann. Für andere Datenformate, z.B. das PLOTCP-Format /4/, muß eine Anpassung durchgeführt werden. Die Dateiart wird durch eine Kennzeichnung im %ALLOC-Kommando angegeben, siehe Abschnitt 2.4.10.

Zum aktuellen Zeitpunkt können Daten in Form von
- Kreisdiagrammen,
- Histogrammen,
- Kurvendiagrammen,
- dreidimensionalen Histogrammen und
- Isoflächen

dargestellt werden. Außerdem hat der Benutzer die Möglichkeit, die Aussagekraft seiner Graphiken mit Hilfe von
- Textbildern

zu verdeutlichen. Weitere Darstellungsarten können leicht in den Präsentationsmodul integriert werden.

Festgelegt wird jede Darstellungsart durch eine Anzahl von Attributen (z.B. Farbe der Säulen eines Histogrammes, Lage des Kreismittelpunktes eines Kreisdiagrammes, Textgröße der Überschrift in einem Bild usw.). Standardwerte für diese Attribute sind in einer Datei abgespeichert, dem VIEW-Workspace.

Das Aufrufen einer bestimmten Darstellung geschieht mittels Kommandoeingabe. Der Präsentationsmodul produziert dann das gewünschte Bild unter Verwendung der Standardwerte. Der Benutzer kann die Attribute mit Hilfe von bestimmten Kommandos verändern und sich einen oder mehrere eigene Workspaces anlegen - dies geschieht wieder mittels Kommandoeingabe -, die dann auch nach der Sitzung erhalten bleiben. Die durch den Präsentationsmodul gewonnenen Bilder können anschließend mit Hilfe des Bildeditors weiterverarbeitet und gespeichert werden.

2.3.2 Die Dateien für die darzustellenden Daten

Die Werte für den Präsentationsmodul, die bildlich dargestellt werden sollen, können in beliebig vielen sequentiellen Fortran-Files gespeichert sein, die durch den E82-Filenamen angesprochen werden.

Unter einem Werteblock versteht man diejenige Menge von Werten, die für eine einzelne Darstellung benötigt wird. Für jede der in 2.3.1 aufgeführten Darstellungsarten gibt es ein bestimmtes Werteblockformat. Jeder Datenfile kann eine beliebige Folge von Werteblöcken in beliebiger Reihenfolge enthalten. Ein bestimmter Werteblock wird durch die Filenummer, den Blocktyp und die laufende Nummer dieses Blocktyps im File identifiziert.

Beispiel eines Datenfiles:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
%STAKFK/IRE6            FISCHER 18/02/83       1
%BBL       8
%PIE   3 %          8
ENERGIEPOLITIK IM KRAFTWERK     BRUTTO-ERZEUGUNG NACH ENERGIE-
TRAEGERN (1981)
      1.74E+01       4.7E+00       1.1E+00       1.0E+00       3.3E+00
      1.29E+01      2.86E+01      3.13E+01
%TEX
KERN-   ENERGIE LAUF-   WASSER  SPEICHERWASSER  ERZEUGTEGASE    HEIZOEL
        ERDGAS          STEIN-  KOHLE   BRAUN-  KOHLE
%EBL
%BBL       6
%SLI   1       3       2
       2       1

DER
GRAPHISCHE EDITOR
E82
%EBL
%EOF
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

Diese Wertedatei besteht aus zwei Werteblöcken : einem PIE- und einem SLIDE-Kennsatz. Sollen z.B. die Daten des PIE-Kennsatzes bildlich dargestellt werden, so muß

1.) das Kommando für ein Kreisdiagramm eingegeben werden,
2.) der Filename der Wertedatei angegeben werden und
3.) eine Blocknummer eingegeben werden, die angibt, welcher PIE-Kennsatz in dieser Datei dargestellt werden soll (im Beispiel 0 oder 1)

Die Werteblöcke und damit auch die gesamten Datenfiles müssen zur Zeit vom Benutzer selbst erstellt (TSO,SPF) oder von einem Programm erzeugt werden. Vorgesehen sind aber E82-Kommandos zum Anlegen solcher Dateien im interaktiven Betrieb. Der genaue Inhalt und das Format der einzelnen Werteblöcke. Weitere Beispiele für Werteblöcke stehen bei den Kommandodefinitionen (2.3.5-2.3.9).

2.3.4 Der Workspace für die Präsentationsgraphik

Für jede der in 2.3.1 aufgeführten Darstellungsarten gibt es eine Anzahl von Standardattributen, die in einer Datei, dem Workspace für die Präsentationsgraphik, gespeichert sind. Beim Aufruf des E82 werden diese Werte eingelesen. Durch Kommandos können die Attribute geändert werden. Für die laufende Sitzung gelten jetzt nicht mehr die Standardwerte, sondern die vom Benutzer selbst definierten. Nach der Sitzung sind diese Werte jedoch wieder verloren, wenn sie nicht durch das Kommando VIEWWRITE auf eine Datei gerettet werden. Man kann beliebig viele eigene Workspaces anlegen und mittels des Kommandos VIEWREAD einlesen (siehe 2.3.14).

2.3.5 Kreisdiagramme

PIE

Das PIE-Kommando erzeugt aus dem in einer Datei gespeicherten Werte-Vektor ein Kreisdiagramm. Die Darstellungsattribute werden aus dem aktuellen VIEW-Workspace entnommen.

Syntax:

PIE fname [bnr]

fname: CHARACTER(8), Filename, siehe 2.4.10

bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste PIE-Datensatz auf der Datei benutzt.

Anwendungsbeispiel:

Eingabe: PIE DVIEW 3 ↑

Effekt : Über den Filename DVIEW wird ein VIEW-File angesprochen, von dem der 3. PIE-Kennsatz bildlich dargestellt werden soll. Der PIE-Kennsatz hat dabei folgendes Format:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
%BBL       5
%PIE   2 %         4
DIE WICHTIGSTEN AUSSAGEN DES    ENERGIEFLUSSBILDES
       50.E+00       30.E+00       15.E+00        5.E+00
%TEX
OEL             KOHLE           GAS             WASSER, KERNENE.
%EBL
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

Auf dem Bildschirm erscheint dann folgendes Bild:

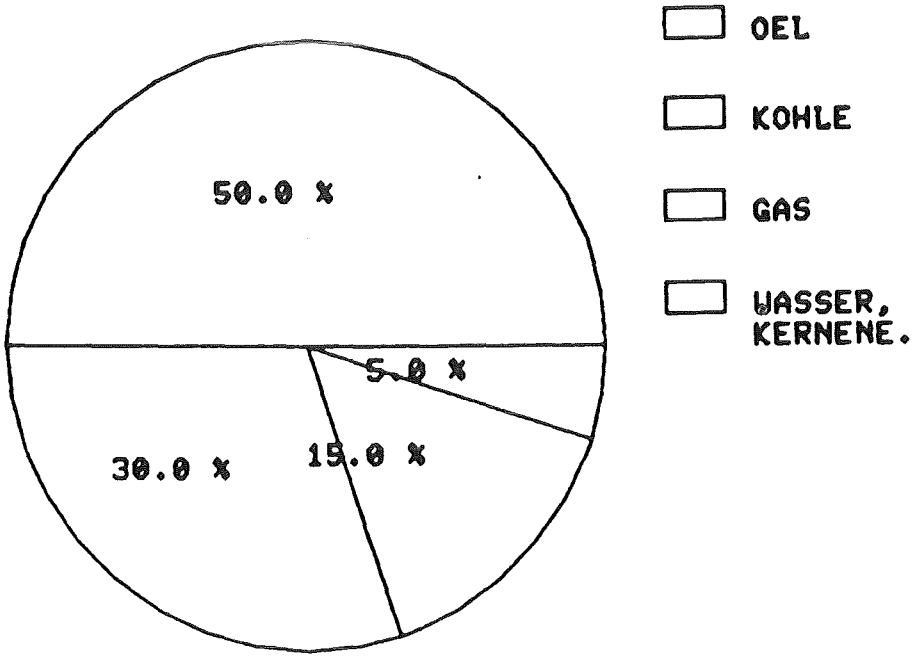


Abbildung 10: Kreisdiagramm C:PIE DVIEW 3

PIEPARMS

Das PIEPARMS-Kommando ändert Darstellungsattribute zur Darstellung von Werten als Kreisdiagramm durch den Präsentationsmodul. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWWRITE-Kommandos auf eine externe Datei gerettet werden.

Syntax:

PIEPARMS

Benutzeranleitung:

Begriffe:

Positionieren des Fadenkreuzes	=	Das am Bildschirm aufgetauchte Fadenkreuz kann mit Hilfe des Steuerknüppels bewegt werden. Ist die gewünschte Position erreicht, so muß die Eingabe durch Drücken einer alphanumerischen Taste beendet werden.
Picken eines Objektes	=	Positionieren des Fadenkreuzes auf ein Objekt und Drücken einer Taste
Texteingabe	=	Eingabe eines Textes mittels alphanumerischer Tastatur und Abschluß der Eingabe durch Drücken der RETURN-Taste

Allgemeines:

Das Kommando PIEPARMS bewirkt den Aufruf einer interaktiv arbeitenden Subroutine, die den Benutzer mittels Menütechnik die Attribute, durch die ein Kreisdiagramm festgelegt wird, verändern lassen. Es gibt dafür drei verschiedene Menüs: Menü LAGE, FARBEN und TEXTATTRIBUTE. Nach Eingabe des Befehls PIEPARMS erscheint unmittelbar das Menü LAGE. Es besitzt, wie auch die beiden anderen Menüs, ein rotes Rechteck am oberen Bildrand, in dem eine Benutzeranweisung steht. Außerdem erscheint ein Fadenkreuz. In jedem Menü gibt es drei blaugrüne Kästchen:

- FARBEN, TEXTATTRIBUTE, ENDE in dem Menü LAGE
- LAGE, TEXTATTRIBUTE, ENDE in dem Menü FARBEN
- LAGE, FARBEN, ENDE in dem Menü TEXTATTRIBUTE

Positioniert man das Fadenkreuz auf das Kästchen ENDE, so wird aus dem Kommando herausgesprungen, positioniert man es auf ein anderes der oben aufgeführten Kästchen, so wird in das entsprechende Menü gesprungen. Dies geschieht dadurch, daß die einzelnen Teilbilder, die ein Menü bestimmen, gelöscht und danach die Teilbilder des neuen Menüs gezeichnet werden.

Menü LAGE:

In diesem Menü kann alles verändert werden, was die bildliche Anordnung betrifft (z.B. Radius, Lage Mittelpunkt, Lage Überschrift usw.). Außer Acht gelassen sind in diesem Menü Farben und Textattribute. Die Anweisung in dem roten Rechteck lautet: "PICKE OBJEKT, DAS VERAENDERT WERDEN SOLL". Mit dem Fadenkreuz kann jetzt ein Objekt angefahren werden, dessen Lage verändert werden soll. Möchte man z.B. den Kreisradius ändern, so positioniert man das Fadenkreuz in dem schwarzen Kästchen mit dem Text "RADIUS". Jetzt wird das rote Rechteck oben im Bild überzeichnet und eine neue Benutzeranweisung erscheint: "PICKE PUNKT. DER ABSTAND ZUM MITTELPUNKT IST DANN DER GEWÜNSCHTE RADIUS." Der Punkt muß angefahren werden, dann wird das entsprechende Teilbild gelöscht und mit dem neuen Radius neu gezeichnet. In dem roten Rechteck erscheint wieder die anfängliche Anweisung "PICKE OBJEKT, DAS VERAENDERT WERDEN SOLL" und das nächste Objekt kann gepickt werden, bzw. in ein anderes Menü kann verzweigt werden.

Menü FARBEN:
In diesem Menü kann alles verändert werden, was Farben betrifft (z.B. Farbe der Überschrift usw.). Außer Acht gelassen sind in diesem Menü die Textattribute, d.h., wenn in einem Kästchen z.B. in gelb der Text "FARBE LEGENDENTEXT" steht, so bedeutet dies, daß die Farbe des Legendentextes gelb ist, aber nicht, daß die Textattribute des Legendentextes mit denen des Textes in dem Kästchen übereinstimmen. Als Anweisung in dem roten Rechteck steht: "PICKE OBJEKT, DESSEN FARBE VERAENDERT WERDEN SOLL." Mit dem Fadenkreuz kann jetzt das Objekt angefahren werden, dessen Farbe verändert werden soll. Möchte man z.B. die Farbe des Legendentextes ändern, so positioniert man das Fadenkreuz auf das entsprechende Kästchen. Jetzt erscheint die Benutzeranweisung "PICKE FARBE". Das Fadenkreuz ist verschwunden und der Cursor steht im linken oberen Eck des Bildes. Verlangt wird nun die Auswahl einer Farbe der Palette mit dem Positionierungskreuz. Ist dies geschehen, so wird das entsprechende Teilbild gelöscht und mit dem veränderten Attribut neu gezeichnet. Die anfängliche Anweisung "PICKE OBJEKT, DESSEN FARBE VERAENDERT WERDEN SOLL" erscheint wieder. Pickt man z.B. ein Legendenkästchen - die Legendenkästchen stehen für die Farbe der Kreissektoren - , so erscheint ebenfalls die Anweisung "PICKE FARBE". Ist die Anweisung ausgeführt, so wird das Teilbild noch nicht neu gezeichnet, sondern es erscheint die Anweisung "PICKE ANDERES KAESTCHEN ODER OBJEKT, DESSEN FARBE VERAENDERT WERDEN SOLL". Jetzt kann ein zweites Legendenkästchen gepickt werden, dann wiederholt sich der beschriebene Vorgang, oder es kann ein anderes Objekt gepickt werden. In diesem Falle wird das Teilbild, zu dem die Legendenkästchen gehören, mit den veränderten Farben neu gezeichnet und in dem roten Rechteck erscheint die Anweisung für den Benutzer, die ihm sagt, was er tun muß, um das von ihm gepickte Objekt zu ändern. Hat er diese Anweisung befolgt, so erscheint wieder der anfängliche Befehl "PICKE OBJEKT, DESSEN FARBE VERAENDERT WERDEN SOLL".

Menü TEXTATTRIBUTE:
In diesem Menü kann alles verändert werden, was Textattribute betrifft, d.h. Zeichensatz, Höhe und Breite von Texten. Außer Acht gelassen sind in diesem Menü die Farben. Damit man die Attribute der verschiedenen Texte erkennen kann, sind weiße Kästchen mit dem Text "AB1" neben die entsprechenden Objekte gezeichnet. Der Text "AB1" ist dabei mit den Originaltextattributen geschrieben. Als Anweisung in dem roten Rechteck steht: "PICKE OBJEKT, DESSEN ATTRIBUTE VERAENDERT WERDEN SOLLEN". Möchte man dies tun, so positioniert man das Fadenkreuz in dem entsprechenden schwarzen Kästchen, nicht in dem weißen (z.B. in dem schwarzen Kästchen mit dem Text "ATTR. UEBERSCHRIFT"). Jetzt erscheint in dem roten Rechteck die Anweisung "PICKE ZEICHENSATZ, HOEHE ODER BREITE". Texthöhe und -breite werden im selben Koordinatensystem angegeben wie die Koordinatenwerte, also in Window-Koordinaten (siehe 2.2.5).

1.Fall: Änderung des Zeichensatzes:
Dazu muß man den gewünschten Zeichensatz in der Zeichensatztabelle (momentan besteht die Zeichensatztabelle aus zwei Zeichensätzen) picken. Das entsprechende Teilbild wird gelöscht, mit dem veränderten Zeichensatz neu gezeichnet und die anfängliche Anweisung erscheint wieder.

2.Fall: Änderung der Texthöhe:
Dazu pickt man den violetten Kasten "HOEHE". Jetzt erscheint in dem roten Rechteck der Text "HOEHE DES UEBERSCHRIFTTEXTES:" und dahinter die aktuelle Höhe. Gleich darauf wird das Rechteck überzeichnet und die neue Anweisung "TIPPE NEUE HÖHE PER TASTATUR EIN" erscheint. Ist die neue Höhe eingegeben, so wird das entsprechende Teilbild gelöscht, dasselbe wird mit veränderter Texthöhe neu gezeichnet und die anfängliche Anweisung erscheint wieder.

3.Fall: Änderung der Textbreite:
Dazu pickt man den violetten Kasten "BREITE". Jetzt erscheint in dem roten Rechteck der Text "BREITE DES UEBERSCHRIFTTEXTES:" und dahinter die aktuelle Breite. Gleich darauf wird das Rechteck überzeichnet und die neue Anweisung "TIPPE NEUE BREITE PER TASTATUR EIN" erscheint. Ist dies geschehen, so wird das entsprechende Teilbild gelöscht, dasselbe wird mit veränderter Textbreite neu gezeichnet und die anfängliche Anweisung erscheint wieder.

Anmerkung: Wird in einem Menü das Fadenkreuz nicht auf ein Objekt positioniert, sondern auf irgendeine undefinierte Stelle, so erscheinen in dem roten Rechteck Fehlernachrichten und der Benutzer wird aufgefordert, seine Eingabe zu wiederholen.

PARMSPIE

Das PARMSPIE-Kommando ändert Attribute, die für die Darstellung von Werten als Kreisdiagramm notwendig sind. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWRITE-Befehls auf eine externe Datei gerettet werden.

Syntax:

```
PARMSPIE [{ BOXES pa pb pc | DBOXES pd | LEGTEX | NOLEGTEX |
            RADIUS pr | SEGTEX | NOSEGTEX | STITLE pu | CENPOINT p0 |
            CSEGTEX pcols | FSEGTEX pfonts | HSEGTEX pheigs |
            WSEGTEX pwidts | CLEGTEX pcoll | FLEGTEX pfontl |
            HLEGTEX pheigl | WLEGTEX pwidtl | Ctitle pcolu |
            FTITLE pfontu | HTITLE pheigu | WTITLE pwidtu |
            CLINE plcol | WLINE plwidt | COLOURS [ picol ]* |
            NUMBC pncol }]*
```

pa,pb,pc:	COORD	, pa: linker unterer Punkt des ersten Legendenkästchens pb: rechter oberer Punkt des ersten Legendenkästchens pc: Anfangspunkt des Legendentextes
pd	: REAL	, Abstand Legendenkästchen + Höhe Kästchen
LEGTEX	:	Legendentexte werden geschrieben
NOLEGTEX:		Legendentexte werden nicht geschrieben
pr	: REAL	, Kreisradius
SEGTEX	:	Segmenttexte werden geschrieben
NOSEGTEX:		Segmenttexte werden nicht geschrieben
pu	: COORD	, Lage Überschrift
p0	: COORD	, Mittelpunkt
pcols	: INTEGER	, Farbe Segmenttext
pfonts	: INTEGER	, Zeichensatz Segmenttext
pheigs	: REAL	, Höhe Segmenttext
pwidts	: REAL	, Breite Segmenttext
pcoll	: INTEGER	, Farbe Legendentext
pfontl	: INTEGER	, Zeichensatz Legendentext
pheigl	: REAL	, Höhe Legendentext
pwidtl	: REAL	, Breite Legendentext
pcolu	: INTEGER	, Farbe Überschrift
pfontu	: INTEGER	, Zeichensatz Überschrift
pheigu	: REAL	, Höhe Überschrift
pwidtu	: REAL	, Breite Überschrift
plcol	: INTEGER	, Farbe Segmentumrandung
plwidt	: REAL	, Strichstärke Segmentumrandung
picol	: INTEGER(100)	, Farben Segmente
pncol	: INTEGER	, Anzahl Farben Segmente

2.3.6 Histogramme

HISTO

Das HISTO-Kommando erzeugt aus dem in einer Datei gespeicherten Werte-Vektor ein Histogramm. Die Darstellungsattribute werden aus dem aktuellen VIEW-Workspace entnommen.

Syntax:

HISTO fname [bnr]

fname: CHARACTER(8), Filename, siehe 2.4.10

bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste HISTO-Datensatz auf der Datei benutzt.

Anwendungsbeispiel:

Eingabe: HISTO DVIEW ↑

Effekt : Über den Filename DVIEW wird ein VIEW-File angesprochen, von dem der erste HISTO-Kennsatz bildlich dargestellt werden soll. Der HISTO-Kennsatz hat dabei folgendes Format:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
%BBL      13
%HIS   2          12        2   .TRUE.              1979
1000 MEGAWATTS                  0.      45.       10
MONTHLY AVERAGES OF MAX. AND    MIN. POWER DEMAND FOR 1979
       41.E+00        39.E+00        37.E+00        32.E+00        29.E+00
       27.E+00        26.E+00        26.E+00        27.E+00        31.E+00
       37.E+00        36.E+00        30.E+00        30.E+00        27.E+00
       23.E+00        21.E+00        18.E+00        17.E+00        17.E+00
       18.E+00        21.E+00        26.E+00        26.E+00
%TEX
   JA       FE       MA       AP       MA       JU       JU       AU       SE
   OC       NO       DE
%LEG
MAX DEMAND      MIN DEMAND
%EBL
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

Auf dem Bildschirm erscheint dann das in Abbildung 11 gezeigte Bild.

HISTOPARMS

Das HISTOPARMS-Kommando ändert Darstellungsattribute zur Darstellung von Werten als Histogramm durch den Präsentationsmodul. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWWRITE-Kommandos auf eine externe Datei gerettet werden.

Syntax:

HISTOPARMS

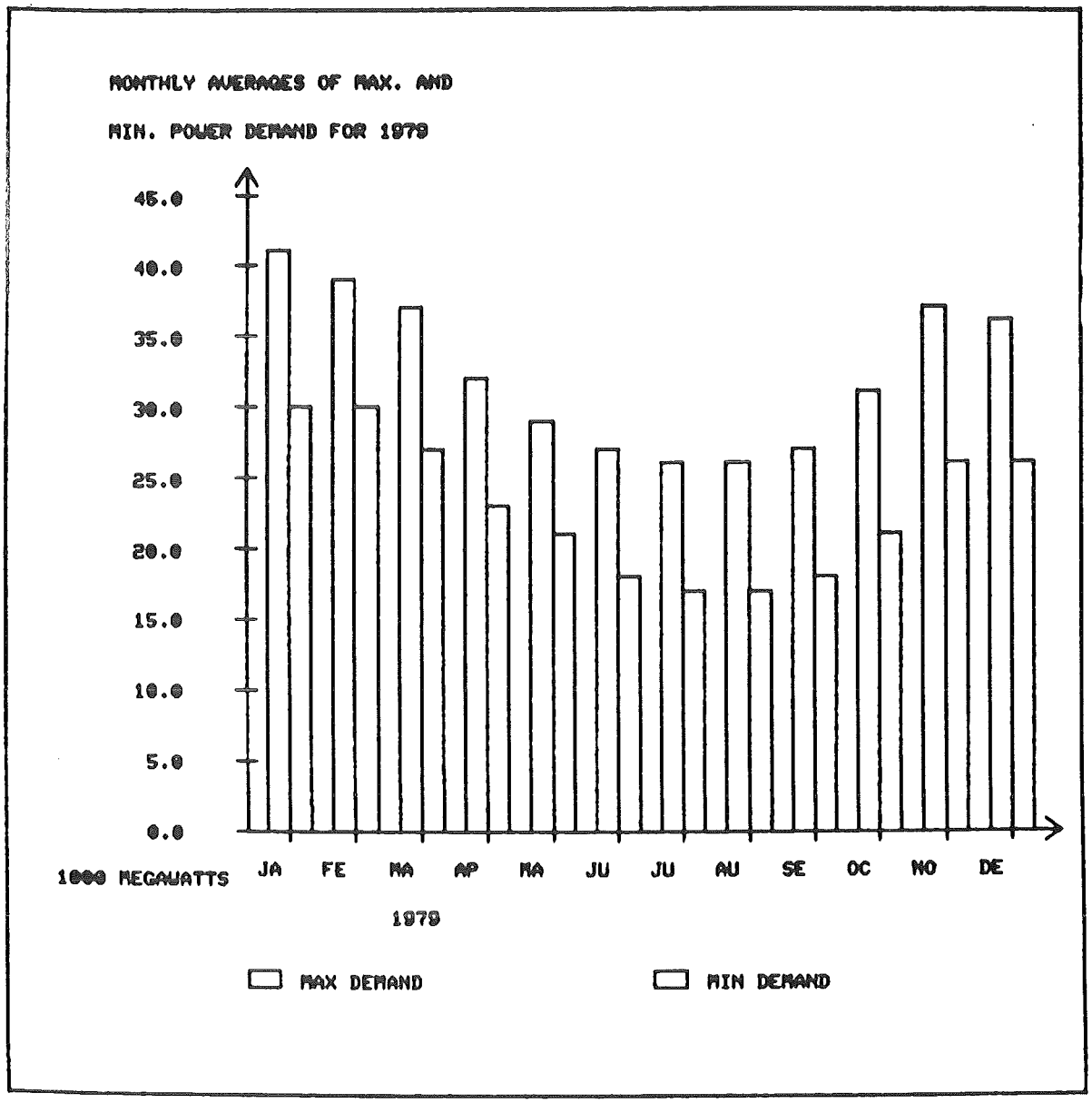


Abbildung 11: Histogramm

Benutzeranleitung:

Begriffe:

Positionieren des Fadenkreuzes = Das am Bildschirm aufgetauchte Fadenkreuz kann mit Hilfe des Steuerknüppels bewegt werden. Ist die gewünschte Position erreicht, so muß die Eingabe durch Drücken einer alphanumerischen Taste beendet werden.

Picken eines Objektes = Positionieren des Fadenkreuzes auf ein Objekt

Texteingabe = Eingabe eines Textes mittels alphanumerischer Tastatur und Abschluß der Eingabe durch Drücken der RETURN-Taste

Allgemeines:
Das Kommando HISTOPARMS bewirkt den Aufruf einer interaktiv arbeitenden Subroutine, die den Benutzer mittels Menütechnik die Attribute, durch die ein Histogramm festgelegt wird, verändern lassen. Es gibt dafür drei verschiedene Menüs: Menü LAGE, FARBEN und TEXTATTRIBUTE. Nach Eingabe des Befehls HISTOPARMS erscheint unmittelbar das Menü LAGE. Es besitzt, wie auch die beiden anderen Menüs ein rotes Rechteck am oberen Bildrand, in dem eine Benutzeranweisung steht. Außerdem erscheint ein Fadenkreuz. In jedem Menü gibt es drei blaugrüne Kästchen:

- FARBEN, TEXTATTRIBUTE, ENDE in dem Menü LAGE
- LAGE, TEXTATTRIBUTE, ENDE in dem Menü FARBEN
- LAGE, FARBEN, ENDE in dem Menü TEXTATTRIBUTE

Positioniert man das Fadenkreuz auf das Kästchen ENDE, so wird aus dem Kommando herausgesprungen, positioniert man es auf ein anderes der oben aufgeführten Kästchen, so wird in das entsprechende Menü gesprungen. Dies geschieht dadurch, daß die einzelnen Teilbider, die ein Menü bestimmen, gelöscht und danach die Teilbilder des neuen Menüs gezeichnet werden.

Menü LAGE:
In diesem Menü kann alles verändert werden, was die bildliche Anordnung betrifft (z.B. Länge der Achsen, Lage der Überschrift, Lage des Achsenkreuzursprungs usw.). Außer Acht gelassen sind in diesem Menü Farben und Textattribute. Die Anweisung in dem roten Rechteck lautet: "PICKE OBJEKT, DAS VERAENDERT WERDEN SOLL". Mit dem Fadenkreuz kann jetzt ein Objekt angefahren werden, dessen Lage verändert werden soll. Möchte man z.B. den Ursprung verschieben, so positioniert man das Fadenkreuz in dem schwarzen Kästchen mit dem Text "URSPRUNG". Jetzt wird das rote Rechteck oben im Bild überzeichnet und eine neue Benutzeranweisung erscheint: "POSITIONIERE NEUE LAGE" Die Anweisung muß ausgeführt werden, dann wird das entsprechende Teilbild gelöscht und mit dem neuen Ursprung neu gezeichnet. In dem roten Rechteck erscheint wieder die anfängliche Anweisung "PICKE OBJEKT, DAS VERAENDERT WERDEN SOLL" und das nächste Objekt kann gepickt werden, bzw. in ein anderes Menü kann verzweigt werden.

Menü FARBEN:
In diesem Menü kann alles verändert werden, was Farben betrifft (z.B. Farbe der Überschrift usw.). Außer Acht gelassen sind in diesem Menü die Textattribute, d.h. ,wenn in einem Kästchen z.B. in gelb der Text "FARBE LEGENDENTEXT" steht, so bedeutet dies, daß die Farbe des Legendentextes gelb ist, aber nicht, daß die Textattribute des Legendentextes mit denen des Textes in dem Kästchen übereinstimmen. Als Anweisung in dem roten Rechteck steht: "PICKE OBJEKT, DESSEN FARBE VERAENDERT WERDEN SOLL." Mit dem Fadenkreuz kann jetzt das Objekt angefahren werden, dessen Farbe verändert werden soll. Möchte man z.B. die Farbe des Legendentextes ändern, so positioniert man das Fadenkreuz auf das entsprechende Kästchen. Jetzt erscheint die Benutzeranweisung "PICKE FARBE". Das Fadenkreuz ist verschwunden und der Cursor steht im linken oberen Eck des Bildes. Verlangt wird nun die Eingabe einer Farbnummer mittels Texteingabe. Ist dies geschehen, so wird das entsprechende Teilbild gelöscht und mit dem veränderten Attribut neu gezeichnet. Die anfängliche Anweisung "PICKE OBJEKT, DESSEN FARBE VERAENDERT WERDEN SOLL" erscheint wieder. Pickt man z.B. ein Legendenkästchen - die Legendenkästchen stehen für die Farbe der Säulen - , so erscheint ebenfalls die Anweisung "PICKE FARBE". Ist die Anweisung ausgeführt, so wird das Teilbild neu gezeichnet und die anfängliche Anweisung erscheint wieder.

Menü TEXTATTRIBUTE:
In diesem Menü kann alles verändert werden, was Textattribute betrifft, d.h. Zeichensatz, Höhe und Breite von Texten. Außer Acht gelassen sind in diesem Menü die Farben. Damit man die Attribute der verschiedenen Texte erkennen kann, sind weiße Kästchen mit dem Text "AB1" neben die entsprechenden Objekte gezeichnet. Der Text "AB1" ist dabei mit den Originaltextattributen geschrieben. Als Anweisung in dem roten Rechteck steht: "PICKE OBJEKT, DESSEN ATTRIBUTE VERAENDERT WERDEN SOLLEN". Möchte man dies tun, so positioniert man das Fadenkreuz in dem entsprechenden schwarzen Kästchen, nicht in dem weißen (z.B. in dem schwarzen Kästchen mit dem Text "ATTR. UEBERSCHRIFT"). Jetzt erscheint in dem roten Rechteck die Anweisung "PICKE ZEICHENSATZ, HOEHE ODER BREITE". 1.Fall: Änderung des Zeichensatzes:
Dazu muß man den gewünschten Zeichensatz in der Zeichensatztabelle (momentan besteht die Zeichensatztabelle aus zwei Zeichensätzen) picken. Das entsprechende Teilbild wird gelöscht, mit dem veränderten Zeichensatz neu gezeichnet und die anfängliche Anweisung erscheint wieder.

2.Fall: Änderung der Texthöhe:
Dazu pickt man den violetten Kasten "HOEHE". Jetzt erscheint in dem roten Rechteck der Text "HOEHE DES UEBERSCHRIFTTEXTES:" und dahinter die aktuelle Höhe. Gleich darauf wird das Rechteck überzeichnet und die neue Anweisung "TIPPE NEUE HOEHE PER TASTATUR EIN" erscheint. Ist dies geschehen, so wird das entsprechende Teilbild gelöscht, dasselbe wird mit veränderter Texthöhe neu gezeichnet und die anfängliche Anweisung erscheint wieder.

3.Fall: Änderung der Textbreite:
Dazu pickt man den violetten Kasten "BREITE". Jetzt erscheint in dem roten Rechteck der Text "BREITE DES UEBERSCHRIFTTEXTES:" und dahinter die aktuelle Breite. Gleich darauf wird das Rechteck überzeichnet und die neue Anweisung "TIPPE NEUE BREITE PER TASTATUR EIN" erscheint. Ist dies geschehen, so wird das entsprechende Teilbild gelöscht, dasselbe wird mit veränderter Textbreite neu gezeichnet und die anfängliche Anweisung erscheint wieder.

Anmerkung: Wird in einem Menü das Fadenkreuz nicht auf ein Objekt positioniert, sondern auf irgendeine undefinierte Stelle, so erscheinen in dem roten Rechteck Fehlernachrichten und der Benutzer wird aufgefordert, seine Eingabe zu wiederholen.

PARMSHISTO
Das PARMSHISTO-Kommando ändert Attribute, die für die Darstellung von Werten als Histogramm notwendig sind. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWRITE-Befehls auf eine externe Datei gerettet werden.

Syntax:

```
PARMSHISTO [{ DCOCOTEX hab1 | DXAXTEXXAX hab2 |
              XAXTEXXCOORD hab2x | DXFIXAX hab3 | DYAXTEXYAX hab4 |
              YAXTEXYCOORD hab4x | DYFIYAX hab5 |
              XCOORDBOXES hlx1 hlx2 | YCOORDBOXES hly |
              LXAX hlxax | LYAX hlyax | LEGTEX | NOLEGTEX | COTEX |
              NOCOTEX | STITLE hu | ORIGINE h0 | CXAX hlcolx |
              LWXAX hlwidx | CYAX hlcoly | LWYAX hlwidy |
              CXAXTEX hcoltx | FXAXTEX hfontx | HXAXTEX hheitx |
              WXAXTEX hwidtx | CYAXTEX hcolty | FYAXTEX hfonty |
              HYAXTEX hheity | WYAXTEX hwidty | CFIXAX hcolzx |
              FFIXAX hfonzx | HFIXAX hheizx | WFIXAX hwidzx |
              CFIYAX hcolzy | FFIYAX hfonzy | HFIYAX hheizy |
              WFIYAX hwidzy | CCOTEX hcols | FCOTEX hfonts |
              HCOTEX hheigs | WCOTEX hwidts | CTITLE hcolu |
              FTITLE hfontu | HTITLE hheigu | WTITLE hwidtu |
              CLEGTEX hcol1 | FLEGTEX hfont1 | HLEGTEX hheigl |
              WLEGTEX hwidt1 | COLOURS [ hicol ]* |
              NUMBC hncol }]*
```

hab1	:	REAL	, Abstand Säulentext - Säule
hab2	:	REAL	, Abstand x-Achsentext - x-Achse
hab2x	:	REAL	, x-Achsentext x-Koordinate
hab3	:	REAL	, Abstand x-Zahlen - x-Achse
hab4	:	REAL	, Abstand y-Achsentext - y-Achse
hab4x	:	REAL	, y-Achsentext y-Koordinate
hab5	:	REAL	, Abstand y-Zahlen - y-Achse
hlx1, hlx2	:	2*REAL	, x-Koordinaten der linken Punkte der ersten beiden Legendenkästchen
hly	:	REAL	, y-Koordinaten der unteren Punkte der ersten beiden Legendenkästchen
hlxax	:	REAL	, Länge der x-Achse
hlyax	:	REAL	, Länge der y-Achse
LEGTEX:			Legendentexte werden geschrieben
NOLEGTEX:			Legendentexte werden nicht geschrieben
COTEX:			Säulentexte werden geschrieben
NOCOTEX:			Säulentexte werden nicht geschrieben
hu	:	COORD	, Lage Überschrift
h0	:	COORD	, Achsenkreuzursprung
hlcolx:		INTEGER	, Farbe x-Achse
hlwidx:		REAL	, Strichstärke x-Achse
hlcoly:		INTEGER	, Farbe y-Achse
hlwidy:		REAL	, Strichstärke y-Achse
hcoltx:		INTEGER	, Farbe x-Achsentext
hfontx:		INTEGER	, Zeichensatz x-Achsentext
hheitx:		REAL	, Höhe x-Achsentext
hwidtx:		REAL	, Breite x-Achsentext
hcolty:		INTEGER	, Farbe y-Achsentext
hfonty:		INTEGER	, Zeichensatz y-Achsentext
hheity:		REAL	, Höhe y-Achsentext
hwidty:		REAL	, Breite y-Achsentext
hcolzx:		INTEGER	, Farbe Zahlen x-Achse
hfonzx:		INTEGER	, Zeichensatz Zahlen x-Achse
hheizx:		REAL	, Höhe Zahlen x-Achse
hwidzx:		REAL	, Breite Zahlen x-Achse
hcolzy:		INTEGER	, Farbe Zahlen y-Achse

```
   hfonzy:  INTEGER      , Zeichensatz Zahlen y-Achse
   hheizy:  REAL         , Höhe Zahlen y-Achse
   hwidzy:  REAL         , Breite Zahlen y-Achse
   hcols :  INTEGER      , Farbe Säulentext
   hfonts:  INTEGER      , Zeichensatz Säulentext
   hheigs:  REAL         , Höhe Säulentext
   hwidts:  REAL         , Breite Säulentext
   hcolu :  INTEGER      , Farbe Überschrift
   hfontu:  INTEGER      , Zeichensatz Überschrift
   hheigu:  REAL         , Höhe Überschrift
   hwidtu:  REAL         , Breite Überschrift
   hcoll :  INTEGER      , Farbe Legendentext
   hfontl:  INTEGER      , Zeichensatz Legendentext
   hheigl:  REAL         , Höhe Legendentext
   hwidtl:  REAL         , Breite Legendentext
   hicol :  INTEGER(30)  , Farben
   hncol :  INTEGER      , Anzahl Farben
```

2.3.7 Kurvendiagramme

DIAGRAM

Das DIAGRAM-Kommando erzeugt aus den in einer Datei gespeicherten Werten ein Kurvendiagramm. Die Darstellungsattribute werden aus dem aktuellen VIEW-Workspace entnommen.

Syntax:

DIAGRAM fname [bnr]

fname: CHARACTER(8), Filename, siehe 2.4.10

bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste DIAGRAM-Datensatz auf der Datei benutzt.

Anwendungsbeispiel:

Eingabe: DIAGRAM DVIEW 2 ↑

Effekt : Über den Filename DVIEW wird ein VIEW-File angesprochen, von dem der 2. DIAGRAM-Kennsatz bildlich dargestellt werden soll. Der DIAGRAM-Kennsatz hat dabei folgendes Format:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
%BBL      15
%DIA   3JAHR           3      12
                              1970.    1981.      12
                                 0.     300.      16
DARSTELLUNG DES VERLAUFS DER    LIZENZVERTRAGSENTWICKLUNG BEI
DER KFK UEBER 11 JAHRE.
        2.E+01       3.1E+01       4.4E+01       5.4E+01       6.8E+01
       8.2E+01       9.3E+01        1.E+02      1.12E+02      1.28E+02
      1.37E+02      1.48E+02       4.7E+01       5.5E+01        7.E+01
       8.4E+01       9.4E+01      1.12E+02       1.3E+02      1.41E+02
      1.48E+02      1.55E+02      1.68E+02       1.8E+02        6.E+01
       7.5E+01       9.1E+01      1.14E+02      1.31E+02      1.49E+02
      1.75E+02      1.86E+02      2.15E+02      2.39E+02      2.68E+02
      2.95E+02
%TEX
ABGESCH.LAUFEND KOM.GEN.
%EBL
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

Auf dem Bildschirm erscheint dann folgendes Bild:

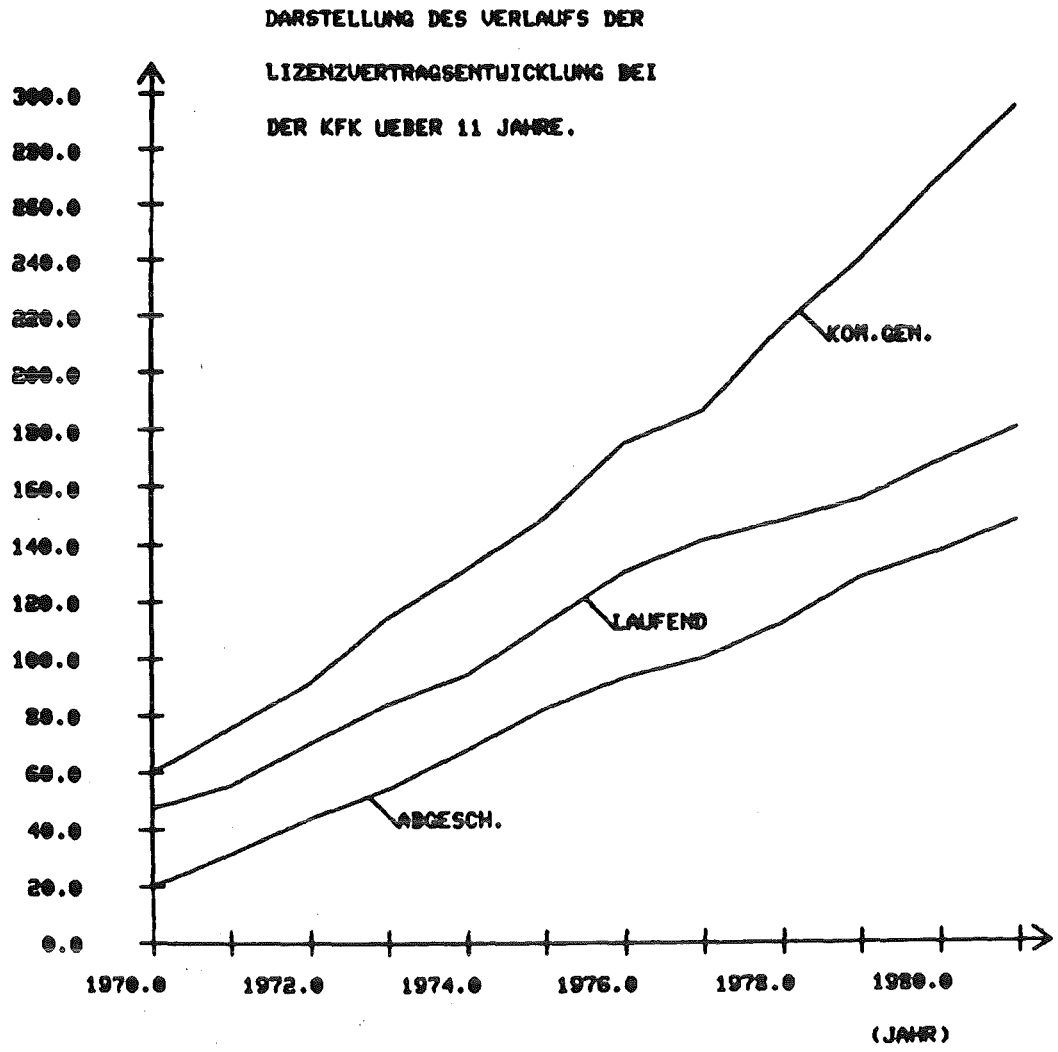


Abbildung 12: Diagramm

PARMSDIA

Das PARMSDIA-Kommando ändert Attribute, die für die Darstellung von Werten als Kurvendiagramm notwendig sind. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWRITE-Befehls auf eine externe Datei gerettet werden.

Syntax:

```
PARMSDIA [{ DCUTEXCU dab1 | DXAXTEXXAX dab2 | XAXTEXXCOORD dab2x |
            DXFIXAX dab3 | DYAXTEXYAX dab4 | YAXTEXYCOORD dab4x |
            DYFIYAX dab5 | CBTWCU | NOCBTWCU | LXAX dlxax |
            LYAX dlyax | CUTEX | NOCUTEX | STITLE du | ORIGINE d0 |
            CXAX dlcolx | LWXAX dlwidx | CYAX dlcoly | LWYAX dlwidy |
            CXAXTEX dcoltx | FXAXTEX dfontx | HXAXTEX dheitx |
            WXAXTEX dwidtx | CYAXTEX dcolty | FYAXTEX dfonty |
            HYAXTEX dheity | WYAXTEX dwidty | CFIXAX dcolzx |
            FFIXAX dfonzx | HFIXAX dheizx | WFIXAX dwidzx |
            CFIYAX dfonzy | FFIYAX dfonzy | HFIYAX dheizy |
            WFIYAX dwidzy | CCUTEX dcolk | FCUTEX dfontk |
            HCUTEX dheigk | WCUTEX dwidtk | CTITLE dcolu |
            FTITLE dfontu | HTITLE dheigu | WTITLE dwidtu |
            CCUS [ dlcol ]* | LWCUS [ dlwid ]* |
            LTCUS [ dnltyp ]* | CBTWCUS [ dicol ]* }]*
```

```
dab1  : REAL          , Abstand Kurventext - Kurve
dab2  : REAL          , Abstand x-Achsentext - x-Achse
dab2x : REAL          , x-Achsentext x-Koordinate
dab3  : REAL          , Abstand Zahlen-x-Achse - x-Achse
dab4  : REAL          , Abstand y-Achsentext - y-Achse
dab4x : REAL          , y-Achsentext y-Koordinate
dab5  : REAL          , Abstand Zahlen-y-Achse - y-Achse
CBTWCU:               , Farben zwischen Kurven
NOCBTWCU:             , keine Farben zwischen Kurven
dlxax : REAL          , Länge der x-Achse
dlyax : REAL          , Länge der y-Achse
CUTEX:                , Kurventexte
NOCUTEX:              , keine Kurventexte
du    : COORD         , Lage Überschrift
d0    : COORD         , Achsenkreuzursprung
dlcolx: INTEGER       , Farbe x-Achse
dlwidx: REAL          , Strichstärke x-Achse
dlcoly: INTEGER       , Farbe y-Achse
dlwidy: REAL          , Strichstärke y-Achse
dcoltx: INTEGER       , Farbe x-Achsentext
dfontx: INTEGER       , Zeichensatz x-Achsentext
dheitx: REAL          , Höhe x-Achsentext
dwidtx: REAL          , Breite x-Achsentext
dcolty: INTEGER       , Farbe y-Achsentext
dfonty: INTEGER       , Zeichensatz y-Achsentext
dheity: REAL          , Höhe y-Achsentext
dwidty: REAL          , Breite y-Achsentext
dcolzx: INTEGER       , Farbe Zahlen x-Achse
dfonzx: INTEGER       , Zeichensatz Zahlen x-Achse
dheizx: REAL          , Höhe Zahlen x-Achse
dwidzx: REAL          , Breite Zahlen x-Achse
dcolzy: INTEGER       , Farbe Zahlen y-Achse
dfonzy: INTEGER       , Zeichensatz Zahlen y-Achse
dheizy: REAL          , Höhe Zahlen y-Achse
dwidzy: REAL          , Breite Zahlen y-Achse
dcolk : INTEGER       , Farbe Kurventext
dfontk: INTEGER       , Zeichensatz Kurventext
dheigk: REAL          , Höhe Kurventext
dwidtk: REAL          , Breite Kurventext
dcolu : INTEGER       , Farbe Überschrift
dfontu: INTEGER       , Zeichensatz Überschrift
dheigu: REAL          , Höhe Überschrift
dwidtu: REAL          , Breite Überschrift
dlcol : INTEGER(10)   , Farben Kurven
dlwid : REAL(10)      , Strichstärken Kurven
dnltyp: INTEGER(10)   , Stricharten Kurven
dicol : INTEGER(11)   , Farben zwischen Kurven
```

2.3.8 Erzeugung von Textvorlagen

SLIDE

Das SLIDE-Kommando erzeugt aus den in einer Datei gespeicherten Werten eine Text-Folie oder ein Dia. Die Darstellungsattribute werden aus dem aktuellen Workspace (in dem unten aufgeführten Beispiel ist es nicht der Standard-Workspace) entnommen.

Syntax:

```
SLIDE fname [bnr]
fname: CHARACTER(8), Filename, siehe 2.4.10
bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an
     gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste
     SLIDE-Datensatz auf der Datei benutzt.
```

Anwendungsbeispiel:
Eingabe: SLIDE DVIEW 2 ↑
Effekt : Über den Filename DVIEW wird ein VIEW-File angesprochen, von dem der 2. SLIDE-Kennsatz bildlich dargestellt werden soll. Der SLIDE-Kennsatz hat dabei folgendes Format:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
%BBL      14
%SLI   1      11       2
       1      10
KFK-FORSCHUNGSPROGRAMM
NACH FORSCHUNGSSCHWERPUNKTEN
PROJEKT SCHNELLER BRUETER (PSB)
TRENNDUESENVERFAHREN (TDV)
FUSIONS- UND TIEFTEMPERATURTECHNIK
PROJEKT WIEDERAUFARBEITUNG UND
ABFALLBEHANDLUNG (PWA)
TIEFLAGERUNG (TLA)
PROJEKT NUCLEARE SICHERHEIT (PNS)
KERNMATERIALUEBERWACHUNG (KMUE)
INNOVATIONEN UND NEUE AUFGABEN (INNA)
GRUNDLAGENFORSCHUNG (GRUND)
%EBL
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

Auf dem Bildschirm erscheint dann folgendes Bild:

KFK-FORSCHUNGSPROGRAMM

NACH FORSCHUNGSSCHWERPUNKTEN

PROJEKT SCHNELLER BRUETER (PSB)
TRENNDUESENVERFAHREN (TDV)
FUSIONS- UND TIEFTEMPERATURTECHNIK
PROJEKT WIEDERAUFARBEITUNG UND
ABFALLBEHANDLUNG (PWA)
TIEFLAGERUNG (TLA)
PROJEKT NUCLEARE SICHERHEIT (PNS)
KERNMATERIALUEBERWACHUNG (KMUE)
INNOVATIONEN UND NEUE AUFGABEN (INNA)
GRUNDLAGENFORSCHUNG (GRUND)

Abbildung 13: Textvorlage

PARMSSLIDE
Das PARMSSLIDE-Kommando ändert Attribute, die für die Darstellung von Werten als Slide notwendig sind. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWRITE-Befehls auf eine externe Datei gerettet werden.

Syntax:

```
PARMSSLIDE [{ STITLE su | TYPEWI [ salign ]* |
              DLINESWI [ szabs ]* | SWI [ st ]* |
              DLINESTITLE szabu | CWI [ scolt ]* |
              FWI [ sfontt ]* | HWI [ sheigt ]* |
              WWI [ swidtt ]* | CTITLE scolu | FTITLE sfontu |
              HTITLE sheigu | WTITLE swidtu }]*
su     : COORD        , Lage Überschrift
salign: INTEGER(10) , Typ des Windows
                       (1: linksbündig
                        2: rechtsbündig
                        3: zentriert )
szabs : REAL(10)     , Zeilenabstand im Window (<0 ==> 2*sheigt)
st     : COORD(10)    , Anfangspunkt für erste Zeile der Windows
szabu : REAL         , Zeilenabstand Überschrift (<0 ==> 2*sheigu)
scolt : INTEGER(10) , Farbe Text in Windows
sfontt: INTEGER(10) , Zeichensatz Text in Windows
sheigt: REAL(10)     , Höhe Text in Windows
swidtt: REAL(10)     , Breite Text in Windows
scolu : INTEGER      , Farbe Überschrift
sfontu: INTEGER      , Zeichensatz Überschrift
sheigu: REAL         , Höhe Überschrift
swidtu: REAL         , Breite Überschrift
```

2.3.9 Dreidimensionale Histogramme

HISTO3D
Das HISTO3D-Kommando erzeugt aus den in einer Datei gespeicherten Werten ein 3D-Histogramm. Die Darstellungsattribute werden aus dem aktuellen VIEW-Workspace entnommen.

Syntax:
HISTO3D fname [bnr]
fname: CHARACTER(8), Filename, siehe 2.4.10
bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste HISTO3D-Datensatz auf der Datei benutzt.

Anwendungsbeispiel:
Eingabe: HISTO3D DVIEW ↑
Effekt : Über den Filename DVIEW wird ein VIEW-File angesprochen, von dem der nächste HISTO3D-Kennsatz bildlich dargestellt werden soll. Der HISTO3D-Kennsatz hat dabei folgendes Format:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
%BBL      12
%H3D   1       5       3JAHR            BEREICH         BIERKONSUM
      BIERKONSUM
      900.E+00       700.E+00       500.E+00       250.E+00       700.E+00
      500.E+00       450.E+00       300.E+00       150.E+00       100.E+00
      200.E+00       120.E+00       120.E+00        70.E+00        70.E+00
%TEX
  900   .                          700      500
          200
%XTE
  1960    1965    1970    1975    1980
%YTE
BAYERN  BADEN   PFALZ
%EBL
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
```

Auf dem Bildschirm erscheint dann folgendes Bild:

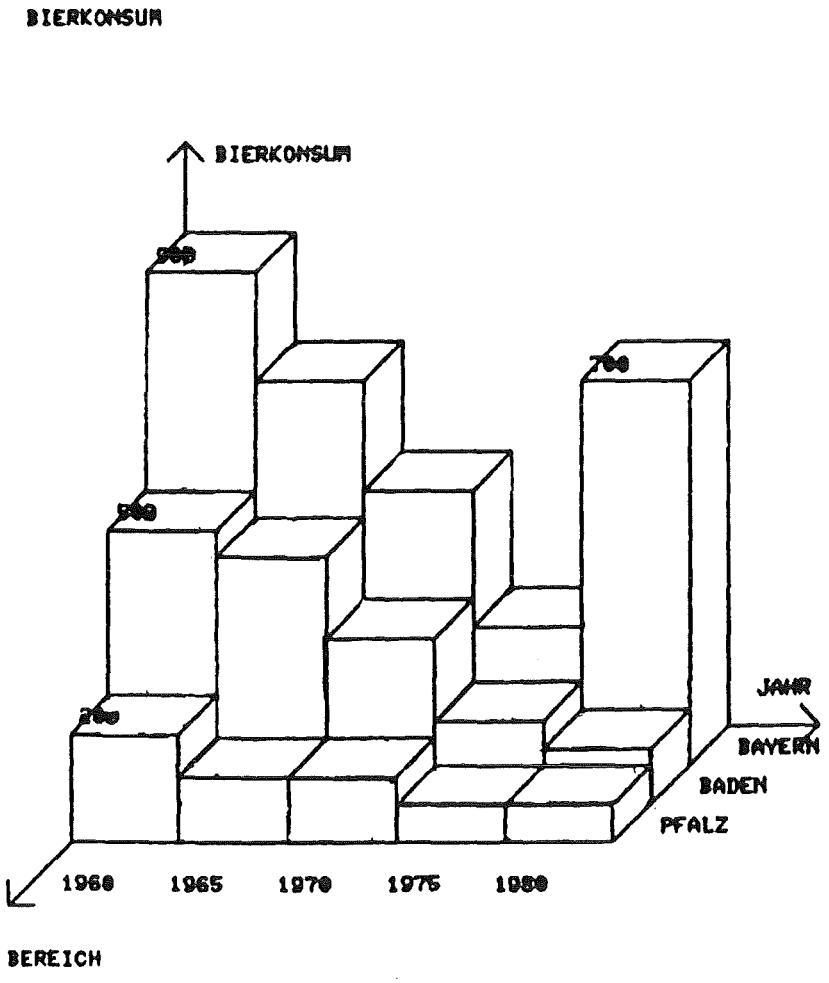


Abbildung 14: 3D-Histogramm

PARMSHISTO3D

Das PARMSHISTO3D-Kommando ändert Attribute, die für die Darstellung von Werten als dreidimensionales Histogramm notwendig sind. Die Attribute sind im VIEW-Workspace gespeichert. Sie können mittels eines VIEWRITE-Befehls auf eine externe Datei gerettet werden.

Syntax:

```
PARMSHISTO3D [{ DCOTEXYCO h3ab1 | DCOTEXXCO h3ab2 |
                DXAXTEXCO h3ab3x | DXAXTEXXAX h3ab3y |
                DYAXTEXYAX h3ab4 | DZAXTEXZAX h3ab5 | LXAX h31xax |
                LYAX h31yax | LZAX h31zax | COTEX | NOCOTEX |
                STITLE h3u | ANGLE h3wink | ORIGINE h30 | CCO h31co |
                LWCO h31wi | CXAX h31cox | LWXAX h31wix |
                CYAX h31coy | LWYAX h31wiy | CZAX h31coz |
                LWZAX h31wiz | CTITLE h3colu | FTITLE h3fonu |
                HTITLE h3heiu | WTITLE h3widu | CXAXTEX h3colx |
                FXAXTEX h3fonx | HXAXTEX h3heix | WXAXTEX h3widx |
                CYAXTEX h3coly | FYAXTEX h3fony | HYAXTEX h3heiy |
                WYAXTEX h3widy | CZAXTEX h3colz | FZAXTEX h3fonz |
                HZAXTEX h3heiz | WZAXTEX h3widz | CCOTEX h3cols |
                FCOTEX h3fons | HCOTEX h3heis | WCOTEX h3wids |
                CCOTEXX h3cosx | FCOTEXX h3fosx | HCOTEXX h3hesx |
                WCOTEXX h3wisx | CCOTEXY h3cosy | FCOTEXY h3fosy |
                HCOTEXY h3hesy | WCOTEXY h3wisy |
                CFRONT [ h3col1 ]* | CSURF [ h3col2 ]* }]*
```

h3ab1	: REAL	, Abstand Text an Säulen in y-Richtung - Säulen
h3ab2	: REAL	, Abstand Text an Säulen in x-Richtung - Säulen
h3ab3x	: REAL	, Abstand x-Achsentext - Säule
h3ab3y	: REAL	, Abstand x-Achsentext - x-Achse
h3ab4	: REAL	, Abstand y-Achsentext - Endpunkt der y-Achse in vertikaler Richtung
h3ab5	: REAL	, Abstand z-Achsentext - Endpunkt der z-Achse in horizontaler Richtung
h31xax	: REAL	, Länge der x-Achse
h31yax	: REAL	, Länge der y-Achse
h31zax	: REAL	, Länge der z-Achse
COTEX	:	Texte auf Säulen werden geschrieben
NOCOTEX	:	Texte auf Säulen werden nicht geschrieben
h3u	: COORD	, Lage Überschrift
h3wink	: REAL	, Neigungswinkel der y-Achse zur Vertikalen (in Gradmaß)
h30	: COORD	, Achsenkreuzursprung
h31co	: INTEGER	, Farbe Säulenumrandung
h31wi	: REAL	, Strichstärke Säulenumrandung
h31cox	: INTEGER	, Farbe x-Achse
h31wix	: REAL	, Strichstärke x-Achse
h31coy	: INTEGER	, Farbe y-Achse
h31wiy	: REAL	, Strichstärke y-Achse
h31coz	: INTEGER	, Farbe z-Achse
h31wiz	: REAL	, Strichstärke z-Achse
h3colu	: INTEGER	, Farbe Überschrift
h3fonu	: INTEGER	, Zeichensatz Überschrift
h3heiu	: REAL	, Höhe Überschrift
h3widu	: REAL	, Breite Überschrift
h3colx	: INTEGER	, Farbe x-Achsentext

```
h3fonx: INTEGER        , Zeichensatz x-Achsentext
h3heix: REAL           , Höhe x-Achsentext
h3widx: REAL           , Breite x-Achsentext
h3coly: INTEGER        , Farbe y-Achsentext
h3fony: INTEGER        , Zeichensatz y-Achsentext
h3heiy: REAL           , Höhe y-Achsentext
h3widy: REAL           , Breite y-Achsentext
h3colz: INTEGER        , Farbe z-Achsentext
h3fonz: INTEGER        , Zeichensatz z-Achsentext
h3heiz: REAL           , Höhe z-Achsentext
h3widz: REAL           , Breite z-Achsentext
h3cols: INTEGER        , Farbe Säulentext
h3fons: INTEGER        , Zeichensatz Säulentext
h3heis: REAL           , Höhe Säulentext
h3wids: REAL           , Breite Säulentext
h3cosx: INTEGER        , Farbe Text an Säulen in x-Richtung
h3fosx: INTEGER        , Zeichensatz Text an Säulen in x-Richtung
h3hesx: REAL           , Höhe Text an Säulen in x-Richtung
h3wisx: REAL           , Breite Text an Säulen in x-Richtung
h3cosy: INTEGER        , Farbe Text an Säulen in y-Richtung
h3fosy: INTEGER        , Zeichensatz Text an Säulen in y-Richtung
h3hesy: REAL           , Höhe Text an Säulen in y-Richtung
h3wisy: REAL           , Breite Text an Säulen in y-Richtung
h3col1: INTEGER(10)    , Farben Vorderfläche
h3col2: INTEGER(10)    , Farben Seiten- und Oberfläche
```

2.3.10 Iso-Flächen

CONTOUR

Das CONTOUR-Kommando erzeugt aus den in einer Datei gespeicherten Werten ein Höhenlinien-Diagramm. Die Werte können auf der Datei in einem regelmäßigen Rechtecknetz, in einem nicht rechteckigen Netz oder in einem Dreicknetz gespeichert sein (zunächst wird das regelmäßige Netz verwirklicht). Die Darstellungsattribute werden aus dem aktuellen VIEW-Workspace entnommen. Abb. 15 zeigt ein Beispiel eines solchen Diagrammes. Die Datei wird über den Filenamen angesprochen.

Syntax:

```
CONTOUR fname [bnr]
fname: CHARACTER(8), Filename, siehe 2.4.10
bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an
     gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste
     3D-Datensatz auf der Datei benutzt.
```

Anwendungsbeispiel:

```
Eingabe: CONTOUR CVIEW 2 ↑
Effekt : Über den Filename CVIEW wird ein VIEW-File angesprochen,
         von dem der nächste 3D-Kennsatz bildlich dargestellt
         werden soll. Der 3D-Kennsatz hat dabei folgendes
         Format:
```

```
%STAKFK/IRE6              BECHLER 22/07/82       1
%BBL
%R3D   1  CM  CM BAR        7        6        5
XACHSE                              0.E+00  15.E+00         4
YACHSE                              0.E+00  10.E+00         3
ZACHSE IN BAR                      -1.E+00   3.E+00         4
BEISPIEL FUER EIN NIVEAU
        1.E+00          1.E+00          1.E+00          2.E+00          1.E+00
        2.E+00          2.E+00
        2.E+00          2.E+00          1.E+00          3.E+00          1.E+00
        1.E+00
       -1.E+00         -1.E+00          0.E+00          1.E+00          2.E+00
       +3.E+00         +0.E+00          3.E+00          1.E+00         -2.E+00
        5.E+00         -2.E+00          6.E+00         -1.E+00          0.E+00
        1.E+00          2.E+00          3.E+00          1.E+00         -7.E+00
       -1.E+00          2.E+00          1.E+00          5.E+00          1.E+00
        3.E+00         -1.E+00          0.E+00          3.E+00          2.E+00
        1.E+00          4.E+00         -1.E+00          6.E+00         -5.E+00
        0.E+00          2.E+00          4.E+00          1.E+00          0.E+00
        5.E+00        1 5E+00
%NIV
       -1.E+00          0.E+00          1.E+00          2.E+00          3.E+00
%EBL
%EOF
```

Auf dem Bildschirm erscheint dann folgendes Bild:

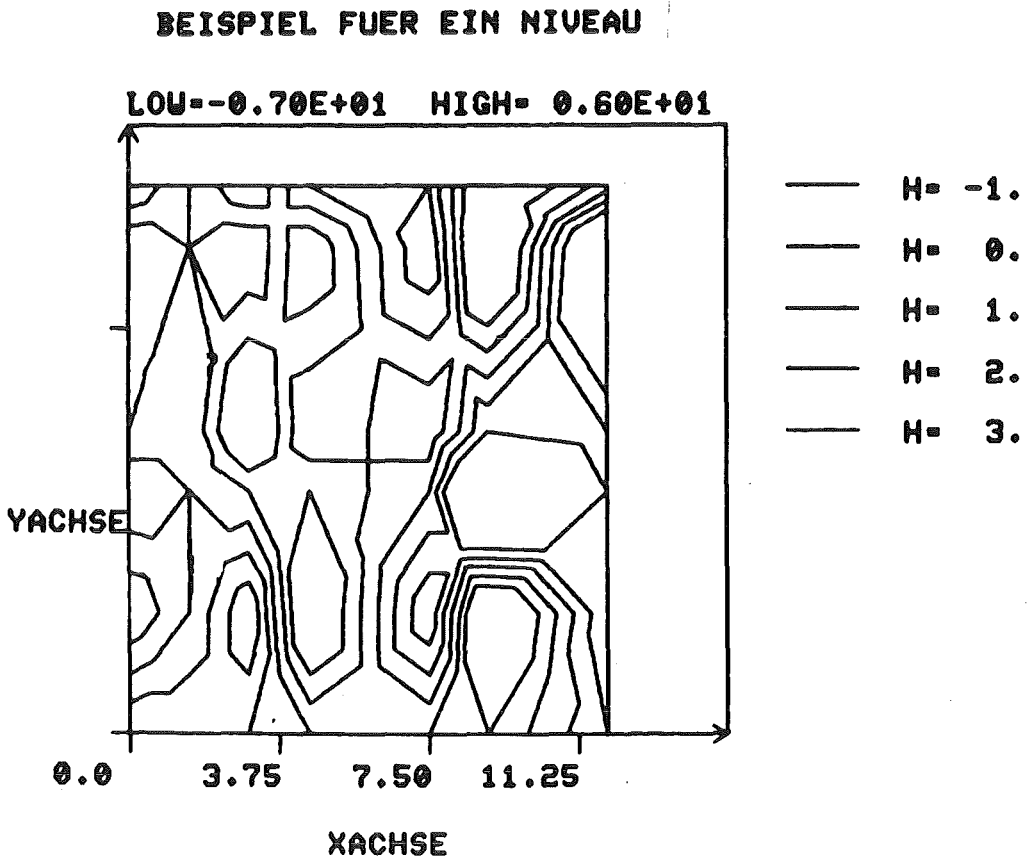


Abbildung 15: Höhenlinien-Diagramm

2.3.11 Schattierte GIPSY-Körper

SHADE
Das SHADE-Kommando gibt eine Pixelmatrix von Farbwerten auf das AED-Terminal aus. Dieses Kommando erzeugt auf anderen Terminals eine Fehlermeldung. Die Pixelmatrix muß von GIPSY /3,7/ erzeugt worden sein. Damit können von GIPSY erzeugte schattierte farbige Körperdarstellungen nachträglich mit dem E82 noch verändert werden. Die Pixelmatrix wird in Form einer Reihe von GKS-Zellmatrizen ausgegeben. Da diese Darstellungen i.a. eine andere Farbtabelle als die normalen Editorfunktionen benötigen, ist das SHADE-Kommando oft in Verbindung mit dem COLOUR FILE Kommando zu benutzen.

Syntax:
SHADE fname [bnr]
fname: CHARACTER(8), Filename, siehe 2.4.10
bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste SHADE-Datensatz auf der Datei benutzt.

2.3.12 Reliefs

RELIEF
Das RELIEF-Kommando erzeugt aus den in einer Datei gespeicherten Werten eine Reliefdarstellung. Die Werte können auf der Datei in einem regelmäßigen Rechtecknetz, in einem nicht rechteckigen Netz oder in einem Dreicknetz gespeichert sein (zunächst wird das regelmäßige Netz verwirklicht). Die Darstellungsattribute werden aus dem aktuellen VIEW-Workspace entnommen. Abb. 16 zeigt ein Beispiel einer Reliefdarstellung. Die Datei wird über den Filenamen angesprochen.

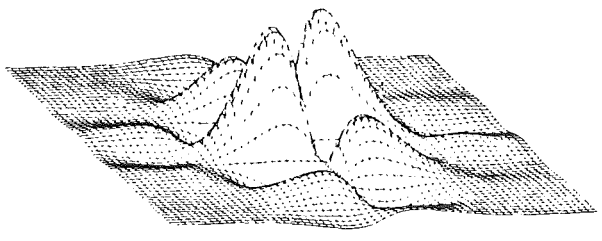


Abb. 16: Relief

Syntax:
RELIEF fname [bnr]
fname: CHARACTER(8), Filename, siehe 2.4.10
bnr: INTEGER, Datensatz-Nummer auf dem File (vom Anfang der Datei an gezählt). Wenn 'bnr' nicht angegeben ist, wird der nächste 3D-Datensatz auf der Datei benutzt.

2.3.13 Speichern und Einlesen des Attribute-Workspaces

VIEWWRITE
Das VIEWWRITE-Kommando rettet einen VIEW-Workspace auf eine externe Datei. Die Datei wird über den Filenamen angesprochen.

Syntax:
VIEWWRITE fname
fname: CHARACTER(8), Filename, siehe 2.4.10

VIEWREAD
Das VIEWREAD-Kommando liest einen VIEW-Workspace von einer externen Datei ein. Die Datei wird über den Filenamen angesprochen.

Syntax:
VIEWREAD fname
fname: CHARACTER(8), Filename, siehe 2.4.10

Beispiel:

```
%ALLOC VIEW1 DSN 'TSO123.VIEW1.DATA' ↑
VIEWREAD VIEW1 ↑
PARMSDIA CBTWCUS 195 200 205 210 215 220 225 ↑
VIEWWRITE VIEW1 ↑
```

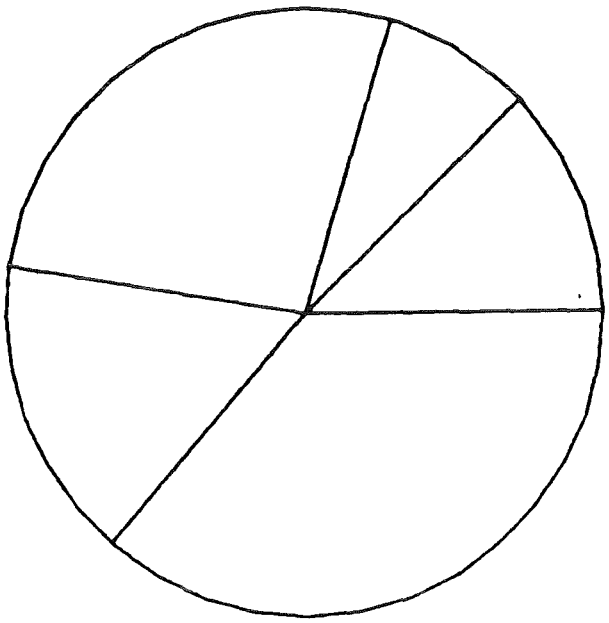
In diesem Beispiel wird eine VIEW-Workspace-Datei mit dem Namen VIEW1 allokiert und danach mit VIEWREAD dieser Workspace eingelesen. Dann wird ein Attribut, nämlich die Farben zwischen den Kurven für die Diagramme, verändert. Der geänderte Workspace wird auf die gleiche Datei zurückgeschrieben.

2.3.14 Die Verwendung von Editier-Kommandos

Bilder, die vom Datenrepräsentationsmodul hergestellt wurden, können vom graphischen Editor weiter behandelt werden, als ob sie von diesem selbst erstellt worden wären:
- Die Bilder können mit weiteren Grundelementen versehen werden (siehe 2.2.3)
- Die Attribute dieser Grundelemente können verändert werden (siehe 2.2.4)
- Die Segmente eines Bildes können transformiert werden (siehe 2.2.5)
- Die so entstandenen Bilder können wie andere Bilder auch verwaltet werden (siehe 2.2.6)

Anwendungsbeispiel:

Eingabe des Benutzers	Wirkung
PARMSPIE NOLEGTEX ↑	
PARMSPIE NOSEGTEX ↑	VIEW-Workspace wird geändert: keine Segment- und keine Legendentexte für Kreisdiagramme
PIE DVIEW ↑	Es erscheint folgendes Bild

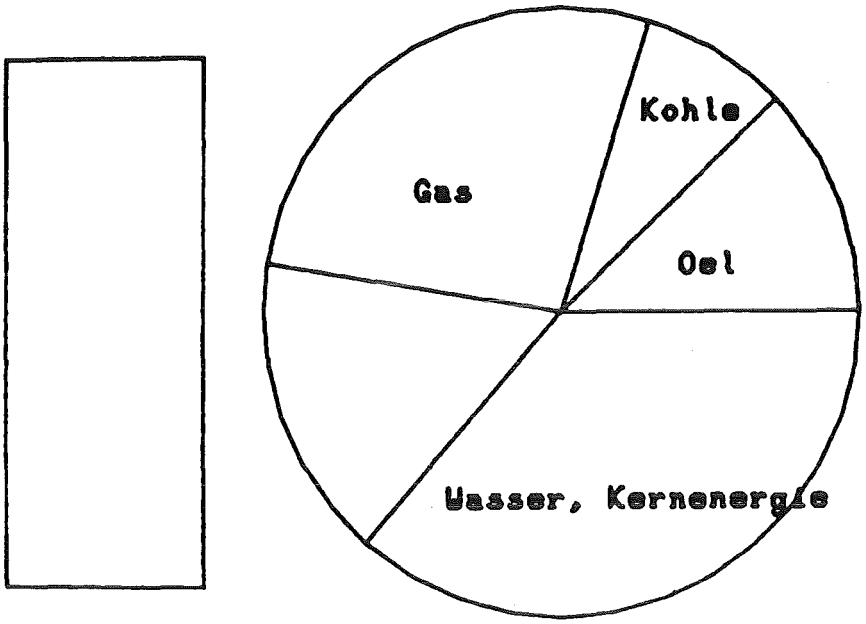


BREAK DOWN INTO COUNTRY GROUPS

OPERATION NUCLEAR POWER PLANTS

Eingabe des Benutzers	Wirkung
TRANSLATE ↑	Das Positionierungskreuz erscheint zum Picken eines Segmentes. Der Kreis wird gepickt. Dann erscheint das Positionierungskreuz zum Verschieben.
TEXT Oel ↑	
Positionieren des Fadenkreuzes	Text 'Oel' im Bild
TEXT Kohle ↑	
Positionieren des Fadenkreuzes	Text 'Kohle' im Bild
TEXT Gas ↑	
Positionieren des Fadenkreuzes	Text 'Gas' im Bild
POLYLINE ↑	
Positionieren des Fadenkreuzes	Polygonzug im Bild
TEXT Wasser, Kernenergie ↑	
Positionieren des Fadenkreuzes	Text 'Wasser, Kernenergie' im Bild

Das Bild hat jetzt folgendes Aussehen:



BREAK DOWN INTO COUNTRY GROUPS

OPERATION NUCLEAR POWER PLANTS

Eingabe des Benutzers	Wirkung
WRITE BDATEI ↑	Das Bild wird auf die Bilddatei mit dem Filenamen BDATEI gespeichert.

2.4 Die Kommandos für den Kommandoprozessor

Der Kommandoprozessor ist der Modul des graphischen Editors E82, der die gesamte Benutzerkommunikation übernimmt. Er benutzt dazu die GKS-Funktionen, im wesentlichen die GKS-Eingabefunktionen. Der Kommandoprozessor verarbeitet zwei Arten von Kommandos: Solche, die für den Kommandoprozessor selbst bestimmt sind, wie z.B. Makrodefinitionen, und Ausführungskommandos, deren Inhalt an das rufende Programm weitergegeben wird. Dazu gehören die Kommandos für den Bildeditor oder den Datenrepräsentationsmodul. Der Name und die Syntax der Kommandos, die der Kommandoprozessor verarbeiten kann, sind in einer Tabelle, dem CMD-Workspace gespeichert. Zur Zeit wird das Hinzufügen neuer Kommandos zu diesem Workspace und das Ändern der Kommandos nicht mit Hilfe des Editors E82 erledigt, sondern durch Anlegen der Workspace-Datei mit dem SPF-Editor. Das Format ist ähnlich dem in dieser Anleitung benutzten Format zum Beschreiben der Syntax der Kommandos.

2.4.1 Der Ablauf der Kommandobearbeitung

Nach der Erledigung eines Auftrages ruft der Editor E82 den Kommando-Prozessor zum Anfordern eines neuen Kommandos und zur Kommandoanalyse. Zunächst wird ein komplettes Kommando eingelesen, i.a. von der Tastatur. Der Kommandoprozessor zerlegt nun das Kommando in seine Bestandteile und sucht den Kommandonamen in einer Tabelle, der Kommandotabelle. Ist das Kommando dort vorhanden, wird das vom Anwender eingegebene Kommando mit der Syntaxbeschreibung für das Kommando verglichen, die ebenfalls in der Kommandotabelle vorhanden ist. Hat der Anwender alle notwendigen Angaben im Kommando gemacht, so wird das entsprechende Programm zur Ausführung des Kommandos aufgerufen und danach das nächste Kommando vom Anwender angefordert. Fehlen jedoch im Kommando Angaben, die notwendig sind, so werden sie vom Anwender über den interaktiven Arbeitsplatz nachgefordert. Das ist natürlich im Stapelbetrieb nicht möglich.

2.4.2 Bestandteile eines Kommandos

Jedes Kommando des E82 besteht aus folgenden Grundbestandteilen, die durch Leerzeichen oder Komma getrennt sind:

- Kommandoname - erstes Element jedes Kommandos
- Integer - ganze Zahlen (INTEGER),
- Real - Gleitkommazahlen (REAL),
- String - Zeichenketten (STRING),
- Koordinatenpaare (x,y) (LOCATOR),
- Segmentnamen (SEGMENT),
- Benennungen (NAME).

Während der Kommandoname und Schlüsselwörter stets so stehen müssen, wie sie in der Kommandodefinition angegeben sind (mit der Ausnahme, daß sie bis auf die signifikanten Buchstaben abkürzbar sind), können die restlichen Parameter in verschiedener Form auftreten: Die Parameter können als Konstante, als Variable, als Makroparameter oder als angeforderte Werte auftreten.

Konstante:
- Der Kommandoname ist immer eine Konstante, und zwar eine Benennung mit der maximalen Länge 32.
- Integer im Format I1 bis I8,
- Real im Format F1.x bis F8.x und E3.x bis E16.x,
- Strings in Hochkomma,
- Koordinaten in Klammern und mit Komma getrennt (real,real),
- Segmentnamen: Buchstaben S% gefolgt von Integer iii, also S%iii.
- Benennungen bestehen aus Buchstaben, Ziffern und den Zeichen '@', '#', '$', '_'. Sie dürfen nicht mit einer Ziffer oder mit '_' anfangen.

Variable:
- Die Variablen müssen mit einem %DECLARE-Kommando vereinbart werden,
- sie müssen mit einem %SET-Kommando einen Wert erhalten haben,
- Variablennamen beginnen stets mit %, danach folgt eine Benennung,
- siehe Kommandos %DECLARE und %SET.

Makroparameter werden beim Makroaufruf und bei der Makrodefinition näher erläutert:
- Makroparameter sind nur in einer Makrodefinition erlaubt (siehe Kommando %MACRO),
- sie müssen in der Makrodefinition vereinbart sein,
- Makroparameter haben die gleiche Form wie Variable,
- Makroparameter sind lokal zu der Makrodefinition, in der sie vereinbart sind.

Nachfordern von Werten:
- Werte werden immer dann nachgefordert, wenn ein nicht optionaler Parameter fehlt. Das Abbrechen der Eingabe (z.B. nach dem letzten Punkt eines POLYLINE) wird durch eine spezielle Eingabe angezeigt. Bei Koordinaten- und Picker-Eingabe geschieht dies durch Drücken der Taste 'E' oder 'Q', bei Texteingabe durch Eingabe des Leerstrings (Drücken der ENTER-Taste).
- Neben der automatischen Nachforderung fehlender Werte kann ein Wert auch explizit nachgefordert werden, wenn statt einer Variablen oder einer Konstanten im Kommando eine spezielle Kennzeichnung steht. Die Kennzeichnungen sind:

INTEGER:	!I
REAL	!R
LOCATOR:	!L
SEGMENT:	!S
STRING:	!C
NAME:	!N

Die Werte werden mit Hilfe der GKS-Eingabefunktionen angefordert. Im Normalfall werden an den Geräten AED und Tektronix LOCATOR-Werte und das Picken von Segmenten mit dem Positionierungskreuz bzw. Fadenkreuz eingelesen, INTEGER-, REAL-, STRING- und NAME-Werte mit der Tastatur. Jedoch kann dies von Kommando zu Kommando unterschiedlich sein, da die Art und Weise der Wertenachforderung Teil der Kommandodifinition ist.

Beispiele für Kommando-Parameter:

```
Konstanten: 1, 23, -10, 3.5, 0.4789, -12345.6, 1e3,-3.585E-5,
            'TEXT', 'abc''d', (3.0,5.7), (0,0), S%1, S%23, S%333,
            NAME123, D@#$_456
Variable:   %I, %X, %ABC, %NAME_1
Anforderung: !I, !R, !L, !S, !C, !N
```

Beispiele für ein Kommando mit unterschiedlichen Parametern:

POLYLINE (2,2) (5,6) ↑
Die Parameter liegen als Konstante vor. Das Kommando ist vollständig definiert.

POLYLINE %P1 %P2 ↑
Die Parameter liegen als Variable vor. Sie müssen in diesem Fall vom Typ LOCATOR sein, vorher deklariert worden sein (Kommando %DECLARE) und einen Wert erhalten haben (Kommando %SET).

POLYLINE !L !L !L ↑
Hier werden genau 3 Punktepositionen über die LOCATOR-Eingabe nachgefordert.

POLYLINE ↑
Da hier notwendige Parameter fehlen, werden sie nachgefordert. Der Kommandoprozessor fordert solange Koordinaten an, bis ein 'E' oder ein 'Q' eingegeben wird.

2.4.3 Workspaces des Kommandoprozessors

Der Kommandoprozessor benutzt zwei wichtige Workspaces: Die Kommandodefinitionstabelle und die Dateiliste. Die Kommandodefinitionstabelle enthält die Kommandodefinition für alle Kommandos des Editors E82, die Makrodefinitionen (siehe Kommando %MACRO), die Funktionstastenbelegungen (siehe Kommando %FKEY) und die Sonderzeichen, die der Kommandoprozessor verwendet (siehe Kommando %SYSTEM). Die Kommandodefinitionstabelle kann mit Hilfe der Kommandos %WORKSPACE READ und %WORKSPACE WRITE gelesen oder geschrieben werden.

Beispiel:

```
%MACRO MY_MACRO ↑
...
...
...
%ENDMACRO ↑
%SYSTEM LINEDEL '\' ↑
%WORKSPACE WRITE FILE2 ↑
```

Hier wird ein neues Makro mit dem Namen MY_MACRO definiert und das Zeichen zum Löschen eines Kommandos auf '\' geändert. Der so geänderte Workspace wird dann auf die Datei mit dem Namen FILE2 gespeichert.

Die Dateiliste enthält den E82-Filenamen, den OS-Datasetnamen, die FORTRAN-File-Nummer und einen Filetyp für alle Dateien, mit denen der Editor E82 arbeiten soll. Die Dateiliste kann mit Hilfe der Kommandos %FILELIST READ und %FILELIST WRITE gelesen oder geschrieben werden.

Beispiel:

```
%FREE DVIEW ↑
%ALLOC DVIEW FNR 66 DSN 'TSOOOO.ABC.FORT' TYPE 22 ↑
%FILELIST WRITE FILE5 ↑
```

Hier wird die Datei mit dem Namen DVIEW aus der Dateiliste gestrichen und dann neu allokiert mit der FORTRAN-File-Nr. 66, dem DS-Namen TSOOOO.ABC.FORT und der Fileart 22 (Präsentationsdaten). Die geänderte Dateiliste wird dann auf die Datei mit dem Namen FILE5 gespeichert.

2.4.4 Kommandos, die nicht von der Tastatur kommen

Kommandos für den Editor E82 kommen zunächst einmal von der Tastatur. Sie können aber auch aus anderen Quellen kommen:

- Alphanumerische Tastatur des interaktiven graphischen Arbeitsplatzes (im interaktiven Betrieb),
- aus einer Primäreingabedatei (für Stapelaufträge),
- aus beliebigen Dateien, die mittels %INCLUDE-Kommandos in die Kommandofolge eingefügt werden können,
- aus Kommandofolgen, die mittels eines %FKEY-Kommandos auf eine Funktionstaste gelegt wurden, dabei ist die Auswahl eines Menüelementes auf Bildschirm oder Tablett gleichwertig mit dem Drücken einer Funktionstaste,
- aus Kommandofolgen, die sich aus der Abarbeitung eines Makros ergeben, das aufgerufen wurde.

Das erste Kommando muß stets von der Tastatur bzw. von der Primäreingabedatei kommen. Von dort kann in die anderen Quellenarten verzweigt werden. Aus jeder Quellenart kann in jede andere Quellenart verzweigt werden. Z.B. kann ein vom Bediener gerufenes Kommando ein %INCLUDE-Kommando enthalten, das eine Kommandodatei einliest, die ihrerseits Makroaufrufe enthält. Am Ende einer Quelle wird stets in die rufende Quelle zurückgekehrt. Lediglich bei schwerwiegenden Fehlern kann der E82/CMD-Prozessor auch sofort auf die oberste Ebene, im interaktiven Betrieb also zur Eingabe von der Tastatur, zurückkehren. Alle Nachforderungen von Werten werden von der Primäreingabe gelesen. Auch wenn Verarbeitungsroutinen den Kommandoprozessor aufrufen, um Werte nachzufordern, werden diese von der Primäreingabe angefordert.

2.4.5 Workspace-Kommandos

%WORKSPACE READ

Das %WORKSPACE READ- Kommando dient zum Laden eines Workspaces des Kommandoprozessors von einer Datei. Der Workspace enthält Makro- und Funktionstastendefinitionen, die Definition der Kommandos, sowie die Systemparameter. Der Standard-Workspace wird beim Initialisieren des Kommandoprozessors durch den E82 eingelesen.

Syntax:

```
%WORKSPACE READ fname
fname: STRING(8), Filename
```

%WORKSPACE WRITE

Das %WORKSPACE WRITE-Kommando dient zum Retten eines Workspaces des Kommandoprozessors auf eine Datei. Der Workspace enthält Makro- und Funktionstastendefinitionen, die Kommandodefinitionen und die Ausführungskommandos, und die Systemparameter (siehe %SYSTEM).

Syntax:
%WORKSPACE WRITE fname
fname: STRING(8), Filename

%FILELIST READ

Das %FILELIST READ-Kommando dient zum Laden der Dateiliste von einer Datei. Die Dateiliste enthält eine Tabelle der vom E82 benutzbaren Dateien. Die Standard-Dateiliste wird beim Initialisieren des Kommandoprozessors durch den E82 eingelesen.

Syntax:
%FILELIST READ fname
fname: STRING(8), Filename

%FILELIST WRITE

Das %FILELIST WRITE-Kommando dient zum Retten der Dateiliste auf eine Datei. Die Dateiliste enthält eine Tabelle der vom E82 benutzbaren Dateien. Die Standard-Dateiliste wird beim Initialisieren des Kommandoprozessors durch den E82 eingelesen.

Syntax:
%FILELIST WRITE fname
fname: STRING(8), Filename

2.4.6 Definition von Makros

%MACRO

Das %MACRO-Kommmando dient zur Makrodefinition. Einem Makronamen wird eine Kommandofolge zugeordnet, die formale Parameter enthalten kann. Beim Makroaufruf werden den formalen Parametern über die Parameterliste aktuelle Werte zugeordnet. Eine Makrodefinition beginnt mit dem Kommando %MACRO und wird mit dem Kommande %ENDMACRO abgeschlossen. Ein Makro wird über einen Namen vom Typ Benennung angesprochen, der auf dem %MACRO-Kommando angegeben werden muß.

Syntax:
%MACRO mname ;
command ;
command ;
...
...
...
%ENDMACRO ;

mname: Benennung
command: Beliebiges Kommando (auch %-Kommandos sind erlaubt).

Makronamen dürfen nicht Kommandonamen sein. Makroparameter werden innerhalb der Makrodefinition durch ein %DECLARE-Kommando mit dem Schlüsselwort MPARM deklariert. Beim Aufruf des Makros werden die angegebenen aktuellen Parameter den in der Makrodefinition deklarierten Parametern positionell zugeordnet.

Beispiel:

```
%MACRO TRI ↑
%DECLARE %P1 %P2 LOCATOR MPARM ↑
%DECLARE %P3 LOCATOR MPARM ↑
POLYLINE %P1 %P2 %P3 %P1 ↑
%ENDMACRO ↑
```

Hier wird ein Makro mit dem Namen TRI definiert, das drei Makroparameter %P1, %P2 und %P3 besitzt.

Makroaufruf

Der Makroaufruf erfolgt über den Makronamen. Daher kann der Makroname nicht der Name eines Ausführungskommandos sein.

Syntax:

```
mname [ parm ]*
mname: NAME, Makroname
parm:  Aktuelle Makroparameter
```

Die aktuellen Parameter werden positionell den formalen Parameter der Makrodefinition zugeordnet, bis die dort vorgegebene maximale Anzahl von Parametern erreicht ist. Noch vorhandene überzählige aktuelle Parameter werden ignoriert. Fehlen Makroparameter beim Aufruf, so werden sie nachgefordert.

Beispiel:

```
TRI ↑
```

Das oben definiert Makro wird aufgerufen. Da die drei Parameter fehlen, werden sie über die Koordinateneingabe nachgefordert, und aus ihren Werten ein Dreieck generiert.

```
TRI (10,10) (4,4) (4,16) ↑
```

Das oben definiert Makro wird aufgerufen. Alle drei Parameter sind beim Makroaufruf vorhanden, aus ihren Werten wird ein Dreieck generiert.

2.4.7 Belegung von Funktionstasten

%FKEY

Das %FKEY-Kommmando dient zum Belegen einer Funktionstaste mit einem Kommando der einem Makro. Gleichwertig mit einer Funktionstaste ist ein Menüfeld. Falls das Gerät keine CHOICE-Eingabe (auch keine simulierte) hat, kann die Tastennummer auch von der Tastatur angefordert werden. Die Tastennummer muß immer angegeben werden. Es wird dann das Standard-Auswählergerät des graphischen Arbeitsplatzes als Funktionstastatur benutzt. Es kann aber auch eine Gerätenummer angegeben werden, wenn z.B. eine Arbeitsplatz ein Tablett mit Menüfeldern und eine Funtktionstastatur besitzt.

Syntax:
%FKEY [CHOICE g_nr] [KEY] taste {macro | command}
g_nr: INTEGER, Gerätenummer des Auswählergerätes
taste: INTEGER, Tastennummer des Auswählergerätes
macro: Beliebiges definiertes Makro
command: Beliebiges Kommando

Beispiel:

%FKEY 5 TRI ↑

Der Taste 5 des Standardauswählergerätes wird das oben definierte Makro TRI zugeordnet.

Funktionstastenaufruf

Aus Gründen der Art der Eingabe, die der E82 benutzt (GKS-REQUEST-Eingabe) reicht es nicht, nur eine Funktionstaste zu drücken. Vielmehr muß die Sequenz

ESC fkey

benutzt werden. D.h. nach Drücken der ESC-Taste muß die Funktionstaste gedrückt werden. Das Drücken der Funktionstaste kann auch durch eine andere Art der CHOICE-Eingabe ersetzt werden (z.B. durch ein Menü). Auf Geräten ohne CHOICE-Eingabe wird die fkey-Nummer über die Tastatur angefordert:

ESC integer.

Auf Geräten ohne ESC-Taste wird eine andere Taste verwendet. Das ESC-Zeichen kann auch mittels eines %SYSTEM-Kommandos geändert werden.

Beispiel:

ESC Taste5

Das Drücken der Taste 5 veranlaßt die Ausführung des Makros TRI, also das Anfordern der drei Parameter und das Erzeugen des Dreiecks.

2.4.8 Einfügen von Kommandodateien

%INCLUDE

Das %INCLUDE-Kommmando dient zum Einfügen von Dateien in den Kommando-Eingabestrom. Die Dateien werden über den Filenamen angesprochen. In einer solchen Datei sind die Kommandos durch ';' zu trennen (oder durch ein anderes Trennzeichen, siehe %SYSTEM). Das Erstellen von solchen Dateien geschieht mit dem SPF- oder TSO-Editor.

Syntax:
%INCLUDE file-name
file-name: CHAR(8), dem Editor bekannte Datei.

2.4.9 Variablendeklaration und Wertezuweisung

%DECLARE

Das %DECLARE-Kommando dient zum Vereinbaren von Variablen, die in Kommandos benutzt werden können. Die Variable können danach und erst, nachdem ihnen ein Wert zugewiesen worden ist (siehe %SET), wie Konstante in den Kommandos benutzt werden.

Syntax:

```
%DECLARE %vname [%vname]*
   { INTEGER | REAL | LOCATOR | STRING(ls) | SEGMENT | NAME(ln) }
   [ { GLOBAL | MPARM } ]
vname: NAME, Name der Variablen
ls:    INTEGER, Länge des Strings (Anzahl Zeichen)
ln:    INTEGER, Länge der Benennung (Anzahl Zeichen)
INTEGER, REAL, LOCATOR, STRING, SEGMENT, NAME: Datentyp
GLOBAL: Diese Vereinbarung gilt auch in allen gerufenen Quellen
MPARM:  Diese Vereinbarung deklariert einen Makroparameter (nur
   gültig in Makrodefinitionen).
```

%SET

Das %SET-Kommando dient zum Zuweisen von Werten an Variable des Kommandoprozessors. Die Variable muß vorher vereinbart sein. Nach der Wertzuweisung kann sie wie eine Konstante als Parameter in Kommandos benutzt werden.

Syntax:

```
%SET %vname = value
vname:   NAME, Variablenname
value: Konstante vom Typ der Variablen oder Variable vom gleichen
   Typ. Bei STRING und NAME wird "value" abgeschnitten, falls der
   Wert mehr Zeichen enthält als die Länge der Variablen %vname.
   Ist "value" kürzer, werden die restlichen Stellen der Variablen
   mit Leerzeichen gefüllt. Falls der Typ nicht übereinstimmt,
   wird eine Konversion durchgeführt nach folgenden Regeln:
```

Typ der Variablen links	Typ des Wertes rechts	Konversion
INTEGER	REAL	INT(value)
INTEGER	LOCATOR	mit %XLOC und %YLOC
INTEGER	STRING, NAME	- (Fehler)
INTEGER	SEGMENT	Segmentnummer
REAL	INTEGER	REAL(value)
REAL	LOCATOR	mit %XLOC und %YLOC
REAL	STRING, NAME	- (Fehler)
REAL	SEGMENT	REAL(Segmentnummer)
LOCATOR	INTEGER	mit %LOC(value1,value2)
LOCATOR	REAL	mit %LOC(value1,value2)
LOCATOR	STRING, NAME	- (Fehler)
LOCATOR	SEGMENT	- (Fehler)

STRING	INTEGER	value im I6-Format
STRING	REAL	value im E14.7-Format
STRING	LOCATOR	value als (x,y) mit x und y im Format E14.7
STRING	SEGMENT	value als S%i mit i im I6-Format
STRING	NAME	value
SEGMENT	INTEGER	Segmentname=value
SEGMENT	REAL	Segmentname=INT(value)
SEGMENT	LOCATOR	- (Fehler)
SEGMENT	STRING, NAME	- (Fehler)
NAME	INTEGER, REAL	- (Fehler)
NAME	LOCATOR, SEGMENT	- (Fehler)
NAME	STRING	value, falls eine Benennung vorliegt, sonst Fehler

%LOCX ist eine Funktion, die die x-Koordinate eines LOCATOR als REAL-Wert liefert,
%LOCY ist eine Funktion, die die y-Koordinate eines LOCATOR als REAL-Wert liefert,
%LOC ist eine Funktion, die einen LOCATOR-Wert liefert, der als x- und y-Koordinate die beiden angegebenen REAL-Werte besitzt.

Beispiel:

```
                                        Erläuterung
%DECLARE %ST1 %ST2 STRING(20) ↑
%DECLARE %L1 LOCATOR ↑
%DECLARE %R1 %R2 REAL ↑
%DECLARE %I INTEGER ↑
%DECLARE %N NAME(3) ↑
%SET %R1 = 3.5 ↑                    %R1 erhält den Wert 3.5
%SET %I = 5 ↑                       %I erhält den Wert 5
%SET %L1 = %LOC(%R1,%I) ↑           %L1 erhält den Wert (3.5,5.0)
%SET %R2 = %LOCY(%L1) ↑             %R2 erhält den Wert 5.0
%SET %ST2 = %R2 ↑                   %ST2 erhält den Wert
                                         ' 5.0000000E+00     '
%SET %ST1 = 'STRING1' ↑             %ST1 erhält den Wert 'STRING1'
%SET %N = %ST1 ↑                    %N erhält den Wert STRING1
```

Anmerkung:
Es ist vorgesehen, in einer späteren Ausbaustufe des Editors E82 statt eines Wertes auf der rechten Seite der Zuweisung beliebige Ausdrücke zuzulassen.

2.4.10 Behandlung von Dateien

%ALLOC

Das %ALLOC-Kommando dient zum Zuordnen eines Filenamens zu einer externen Datei. Alle Dateien werden in der Dateiliste des Komamndoprozessors geführt. Das %ALLOC-Kommando kann einem E82-Filenamen einen vorallokierten FORTRAN-File über die FORTRAN-File-Nummer zuordnen. Eine Datei kann aber auch dynamisch allokiert werden. Dann muß ein externer Datensatzname (OS-Dataset-Name, z.B. TSO567.DATEN.DATA') zusätzlich angegeben werden. Außerdem kann das %ALLOC-Kommando dazu benutzt werden, der Datei eine zusätzliche Kennzeichnung zu geben. Diese Kennzeichnung ist eine Integerzahl > 0. Alle Routinen des E82

sprechen Files nur über die Dateiliste an, d.h. alle Files, die in Kommandos angesprochen werden, müssen in der Dateiliste vorhanden sein. Dies kann dadurch geschehen, daß sie vom Dateilisten-Workspace nach dem Aufruf des E82 eingelesen und dann automatisch allokiert werden. Alle Standarddateien, die der E82 und das GKS benötigen, werden so bereitgestellt. Das %ALLOC-Kommando dient zum Einfügen einer neuen Datei in die Dateiliste und zum Eröffnen der Datei. Das %FREE-Kommando schließt eine Datei und streicht sie aus der Dateiliste. Das %REWIND-Kommando dient zum Positionieren des Lese- oder Schreibzeigers auf den Anfang der Datei.

Syntax:

```
%ALLOC fname [ FNR fnr ] [ DSN dsname ] [ TYPE ftype ]
fname: STRING(8), Filename
fnr:   INTEGER, FORTRAN-File-Nummer
dsname: STRING(32), externer Datasetname
ftype: INTEGER, File-Typ
```

Folgende File-Typen sind zur Zeit möglich:

```
 0: nicht spezifiziert
 1: GKS-Trace-Input-File
 2: Print-Output-File
 3: Print-Output-File, Ausgabe unterdrückt
 4: Device Description Table, GKS
 5: Temporary Segment Store
 6: Font-Input-File GKS
 7: E82-Common-Datei
 8: Datei für die Dateiliste
 9: Workspace des Kommandoprozessors
10: E82-Message-Table-Input
11: %INCLUDE-File
12: GKS-Protokolldatei
13: Eingabedatei des E82 für Stapelbetrieb
14: Verbindungsfile für den graphischen Arbeitsplatz
21: PICT-Workspace-Datei
22: aktuelle Bilddatei
23: Bilddatei nur Eingabe
24: Bilddatei nur Ausgabe
25: Bilddatei Aus-und Eingabe
26: AGF-Plotfile
31: VIEW-Workspace-Datei
32: View-Daten im Standardformat
33: PLOTCP-Datei
```

Für den Anwender sind davon i.a. lediglich die Typen 0, 11 und >20 interessant, alle anderen werden im Normalfall nach dem Aufruf des E82 automatisch allokiert. Die Angabe eines Typs erhöht die Sicherheit gegen unbeabsichtigtes Überschreiben einer Datei. Z.B. wird eine Datei vom Typ 23 (Bilddatei Eingabe) vom E82 nicht als Ausgabe-Bilddatei oder als Workspace-Datei benutzt. Dagegen werden Dateien ohne Typ (TYP=0) überall ohne Überprüfung akzeptiert.

Beispiel für eine Dateiliste:

E82-Filename	FNR	TYP	DS-Name
TRACPARM	1	1	TSO147.E82.DATA(NOTRACE)
TRACCOMM	2	3	0
FORTINP	5	2	0
FORTMESS	6	2	0
WSCMD	10	9	TSO147.E82.DATA(FECWSC)
FONT	12	6	TSO147.E82.DATA(GKFONT)
MESSTAB	13	10	TSO147.E82.DATA(MESSAGE)
GKSMESS	14	2	0
E82MESS	14	2	0
E82TRACE	14	2	0
MESSFILE	14	2	0
E82COMM	17	7	TSO147.E82.DATA(FE82COM)
GKSTRACE	20	2	0
ECWSF	21	8	TSO147.E82.DATA(FECWSF)
MFTEMP	22	22	0
WSPICT	24	21	TSO147.E82.DATA(FEPWS)
MFOUT	25	25	0
TABAED	30	4	TSO147.E82.DATA(TABAED)
SEGSTORE	31	5	TSO147.SEGSTORE.DATA
WSVIEW	32	31	TSO147.E82.DATA(FEVWS)
TESTMESS	50	2	0

Beispiel:

```
%ALLOC F1 FNR 77 DSN 'TSO999.ABC.DATA' TYPE 11 ↑
```

Der File mit dem Namen F1 wird allokiert und in die Dateiliste aufgenommen. Die Fortran-File-Nummer ist 77, der DS-Name ist TSO999.ABC.DATA und der Dateityp ist 11.

```
%ALLOC F2 DSN 'TSO999.XXX.DATA' ↑
```

Der File mit dem Namen F2 wird allokiert und in die Dateiliste aufgenommen. Da die Fortran-Nummer nicht angegeben ist, sucht der Kommandoprozessor eine freie Nummmer (von 99 rückwärts). Der Dateityp ist ebenfalls nicht angegeben, daher wird der Typ 0 benutzt.

```
%ALLOC F3 FNR 87 ↑
```

Der File mit dem Namen F3 wird eröffnet und in die Dateiliste aufgenommen. Da der DS-Name nicht angegeben ist, muß der Fortran-File mit der Nummer 87 schon vor Aufruf des E82 allokiert worden sein (z.B. mit dem TSO-Kommando ALLOC F(FT87F001) ...).

Nach der Ausführung der drei %ALLOC-Kommandos sind folgende drei Einträge in die Dateiliste neu aufgenommen worden:

```
F1        77  11   TSO999.ABC.DATA
F2        99   0   TSO999.XXX.DATA
F3        87   0   0
```

%FREE

Das %FREE-Kommando löscht die Zuordnung eines Filenamens zu einer externen Datei und streicht die Datei aus der Dateiliste.

Syntax:

```
%FREE fname
fname: STRING(8), Filename
```

%REWIND

Das %REWIND-Kommando positioniert den Lese- oder Schreibzeiger auf den Anfang der Datei. D.h. nach dem %REWIND wird ein folgendes Lesen oder Schreiben der Datei von Anfang an erfolgen. Beim Schreiben wird dabei der alte Inhalt der Datei überschrieben.

Syntax:

```
%REWIND fname
fname: STRING(8), Filename
```

2.4.11 Trace- und Help-Kommandos

%LIST

Das %LIST-Kommando dient zum Auflisten von Systemdaten des Kommandoprozessors. Makro- und Funktionstastendefinitionen, die verfügbaren Kommandos, die Dateiliste, die Systemparameter oder der Inhalt des gesamten Workspaces können auf eine Datei oder auf den graphischen Arbeitsplatz ausgegeben werden.

Syntax:

```
%LIST
   {
   MACRO mname |
   FKEY  keynr |
   COMMAND cname |
   COMLIST [ ALL ] |
   FILELIST |
   WORKSPACE |
   SYSTEM
   }
   [FILE fname ]

mname: NAME, Makroname
keynr: INTEGER, Funktionstasten-Nummer
mname: NAME, Kommandoname
COMMAND: Definition eines Kommandos
COMLIST: Liste aller Kommandos (nur Namen oder auch Definitionen)
ALL: Liste mit den Definitionen der Kommandos
FILELIST: Dateiliste
```

```
   WORKSPACE: Übersicht über den gesamten Kommando-Workspace
   SYSTEM: Systemparameter des E82/CMD-Prozessors.
   FILE: Ausgabe auf eine Datei
   fname: STRING(8), Filename
```

Beispiel:

```
   %LIST MACRO TRI ↑
   %LIST COMMAND POLYLINE ↑
   %LIST FILELIST FILE E82MESS ↑
   %LIST COMLIST FILE E82MESS ↑
```

%SYSTEM

Das %SYSTEM-Kommando dient zum Ändern von E82/CMD-Systemparametern.

Syntax:

```
   %SYSTEM
      {
      CMDMARK 'cmdmarker' |
      REQUEST 'requestmarker' |
      TERMINATOR 'terminator' |
      ESCMARK 'esc-marker' |
      LINEDEL 'line-delete' |
      CHARDEL 'char-delete'
      }

      cmdmarker: Zeichen für CMD-Kommandos, Standard: '%',
      requestmarker: Zeichen für Anforderung von Eingabe,
        Standard: '!',
      terminator: Zeichen zum Abschließen von Kommandos,
        Standard: ';',
      esc-marker: Zeichen zum Umschalten auf Funktionstasten,
        Standard: ESC.
      line-delete: Zeichen zum Löschen der aktuellen Zeile
        Standard: '<'
      char-delete: Zeichen zum Löschen von Zeichen
        Standard: Backspace (CNTL H auf dem AED-Terminal).
```

%TRACE

Das %TRACE-Kommando schaltet einen Programmaufruf-Trace ein oder aus. Der Trace wird auf die Datei mit dem Filenamen E82TRACE ausgegeben.

Syntax:

```
   %TRACE [ { ON | OFF } ]
   ON: Trace wird eingeschaltet
   OFF: Trace wird ausgeschaltet
 Wird nur %TRACE eingegeben, wird der Trace  eingeschaltet.
```

%DUMP

Das %DUMP-Kommando gibt den Inhalt sämtlicher Workspaces des E82 auf eine Datei aus. Wenn keine Datei angegeben wird, wird der Dump auf die Datei mit dem Filenamen E82TRACE ausgegeben.

Syntax:

%DUMP [FILE fname]
fname: STRING(8), Name der Datei für die Liste der Workspaces

%HELP
Das %HELP-Kommando liefert Informationen über den Zustand des Editors E82 und über die Kommandos.

Syntax:

%HELP [{COMMANDS | MACROS | FKEYS | WORKSPACES }]

%HELP alleine liefert in Situationen, in denen der E82 vom Anwender eine bestimmte Aktion erwartet, eine Erläuterung dieser Situation für den Anwender. Erwartet der E82 eine Kommandoeingabe, erfolgt ein Hinweis auf das erweiterte %HELP-Kommando. %HELP COMMANDS listet alle Kommandos und gibt einen Hinweis auf das %LIST-Kommando, %HELP MACROS und %HELP FKEYS veranlassen entsprechende Informationen. %HELP WORKSPACES listet alle Workspaces und die zugehörigen Kommandos.

Beispiel:

TEXT ↑
Der E82 promptet mit "TEXT:"
Der Anwender antwortet mit %HELP ↑
Der E82 gibt aus:
"Enter a TEXT String via the Keyboard and hit the RETURN-Key"

3. EIN VOLLSTÄNDIGES BEISPIEL

Hier soll ein vollständiges Beispiel dazu dienen, einen neuen Anwender mit der Arbeitsweise des E82 vertraut zu machen. Nach der Devise: "Probieren ist besser als Studieren" sollte der neue E82-Anwender dieses Beispiel zuerst einmal Schritt für Schritt nachvollziehen. Es wird in diesem Beispiel vorausgesetzt, daß die Standard-E82-Dateien den Prefix 'TSO147' haben. Bei anderen Installationen ist dieser Prefix entsprechend zu ändern.

Einschalten ins TSO:
LOGON TSO... PROC(F) SIZE(1024) ↑
Beachte: Der Editor E82 braucht 1024K Arbeitsspeicher.

Aufruf des E82
EXEC 'TSO147.E82.CLIST(E82)' für das AED-Terminal oder
EXEC 'TSO147.E82.CLIST(E82TEK)' für das Tektronix-Terminal
Nach einiger Zeit erfolgt ein dreifacher Piepston, der anzeigt, daß das GKS jetzt initialisiert wird, danach wird der Bildschirm mit der Hintergrundfarbe gefüllt (am AED-Terminal), schließlich meldet sich der E82 mit:

```
C:
```

Dies ist die Aufforderung, ein Kommando einzugeben. Wir wollen uns ein Histogramm anschauen, eine Datei mit Testbeispielen ist im Member DVIEW in TSO147.E82.DATA vorhanden. Diese Testdatei enthält folgende Datensätze:

Anmerkung:
In der Spalte "WORKSPACE" steht ein *, wenn es sich um den Standard-Workspace handelt (E82-Filename WSVIEW, Member FEVWS in TSO147.E82.DATA). Wir benutzen den Standard-Workspace, können also direkt HISTO aufrufen.

ART		BESCHREIBUNG	WORKSPACE
PIE	1	BREAK DOWN INTO COUNTRY-GROUPS	*
	2	BREAK DOWN INTO REACTOR TYPES	*
	3	DIE WICHTIGSTEN AUSSAGEN DES ENERGIEFLUSSBILDES	*
	4	ABFAELLE AUS DEM PRIVATEN BEREICH	*
	5	ENERGIEPOLITIK IM KRAFTWERK (1971)	*
	6	ENERGIEPOLITIK IM KRAFTWERK (1981)	*
HISTO	1	MONTHLY AVERAGES OF MAX.AND.MIN POWER DEMAND FOR 1979	*
	2	" (UEBEREINANDER)	*
	3	ENERGIEVERBRAUCH (NEBENEINANDER)	*
	4	" (UEBEREINANDER)	*
	5	BENZINVERBRAUCH 1981/82	*
	6	KFK-FORSCHUNGSPROGRAMM UEBERSICHT:INVESTITIONEN (1982)	*
	7	KFK-FORSCHUNGSPROGRAMM UEBERSICHT:PERSONAL (1982)	*
	8	DIE BESCHAEFTIGTEN DER GFK	*
DIAGRAM	1	ENERGIEVERBRAUCH	*
	2	DARSTELLUNG DER LIZENZVERTRAGSENTWICKLUNG BEI DER KFK UEBER 11 JAHRE	*
	3	DARSTELLUNG DER FUNKTIONEN F(X)=LN X, G(X)=4/X UND H(X)=-0.25*(X-5)**2+4	*

SLIDE	1	KFK-FORSCHUNGSPROGRAMM	WSVIEW2
	2	DER GRAPHISCHE EDITOR	WSVIEW3
	3	MITARBEITER	WSVIEW4
	4	ALLGEMEINES UEBER DEN E82	WSVIEW5
HISTO3D	1	KONSUM	*
	2	ENERGIEVERBRAUCH	*
	3	KFK-FORSCHUNGSPROGRAMM UEBERSICHT UEBER MITTELEINSATZ	*

Allokieren der Datendatei mit den Histogrammwerten:
%ALLOC DFILE DSN 'TSO147.E82.DATA(DVIEW)' ↑
Mit Hilfe des Namens DFILE können wir nun die Daten als Histogramm darstellen:
Histogramm:
C: HISTO DFILE 2 ↑
Da wir die Histogramm-Nummer 2 angegeben haben, wird nun der zweite Histogramm-Datensatz von der Datei DFILE dargestellt.
Wir erhalten folgendes Bild:

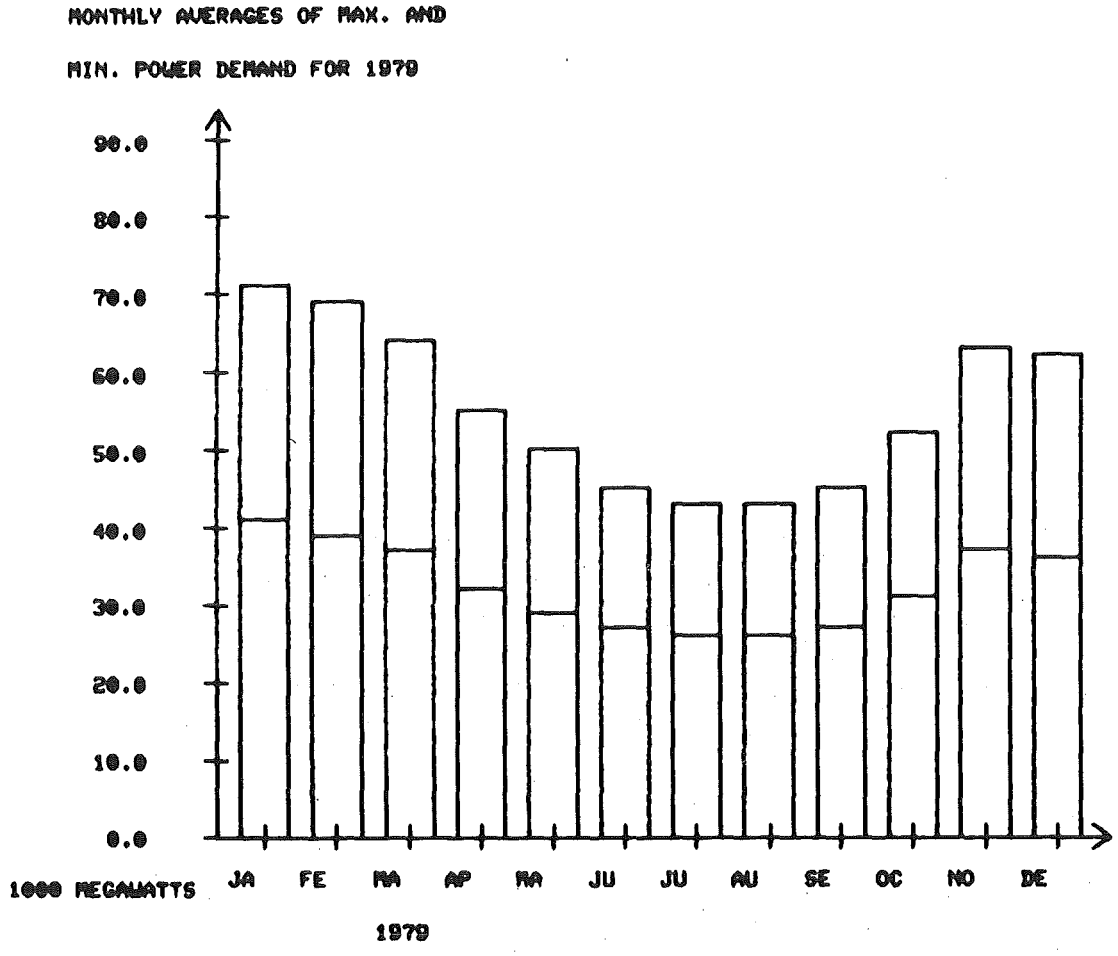


Abbildung 18: Histogramm

Nachdem wir nun das erste Bild auf dem Schirm haben, einige wichtige Dinge zur Kommandoeingabe:
a) Alle Kommandos können auch mit Kleinbuchstaben eingegeben werden.

b) Am AED-Terminal verschwindet manchmal die Aufforderung C:. Dann ist die Taste PAN zu drücken, der Steuerknüppel nach links vorn zu bewegen und durch erneutes Drücken der PAN-Taste das "Pannen" oder "Schwenken" zu beenden.
c) Falls man sich vertippt hat, kann das Kommando dadurch korrigiert werden, daß man um soviel Stellen, wie man ändern will, zurückgeht. Das geschieht auf der Tektronix mit der BACKSPACE-Taste und auf dem AED-Terminal durch gleichzeitiges Drücken von CNTL und H.
d) Will man das ganze Kommando löschen, so geschieht dies durch das Zeichen '<' irgendwo im Kommando.

Wir wollen nun in das Bild einige Rechtecke mit Text und verbindende Linien einbringen. Damit uns die Rechtecke und Linien schön waagerecht und senkrecht gelingen, setzen wir für die folgende Koordinateneingabe ein Raster:

C: GRID 1 1 ↑
Damit werden alle Koordinatenwerte auf ganze Zahlen gezwungen. Die Standardbildausdehnung ist 20 x 20 (cm). Nun bringen wir zunächst ein gefülltes Rechteck rechts oben ins Bild:

C: RECTANGLE (16,17) (19,19) AREA ↑
(16,17) ist der linke untere Eckpunkt des Rechteckes, (19,19) der rechte obere. AREA bedeutet, daß das Rechteck gefüllt werden soll (nur auf dem AED-Terminal). Das Bild sieht nun so aus:

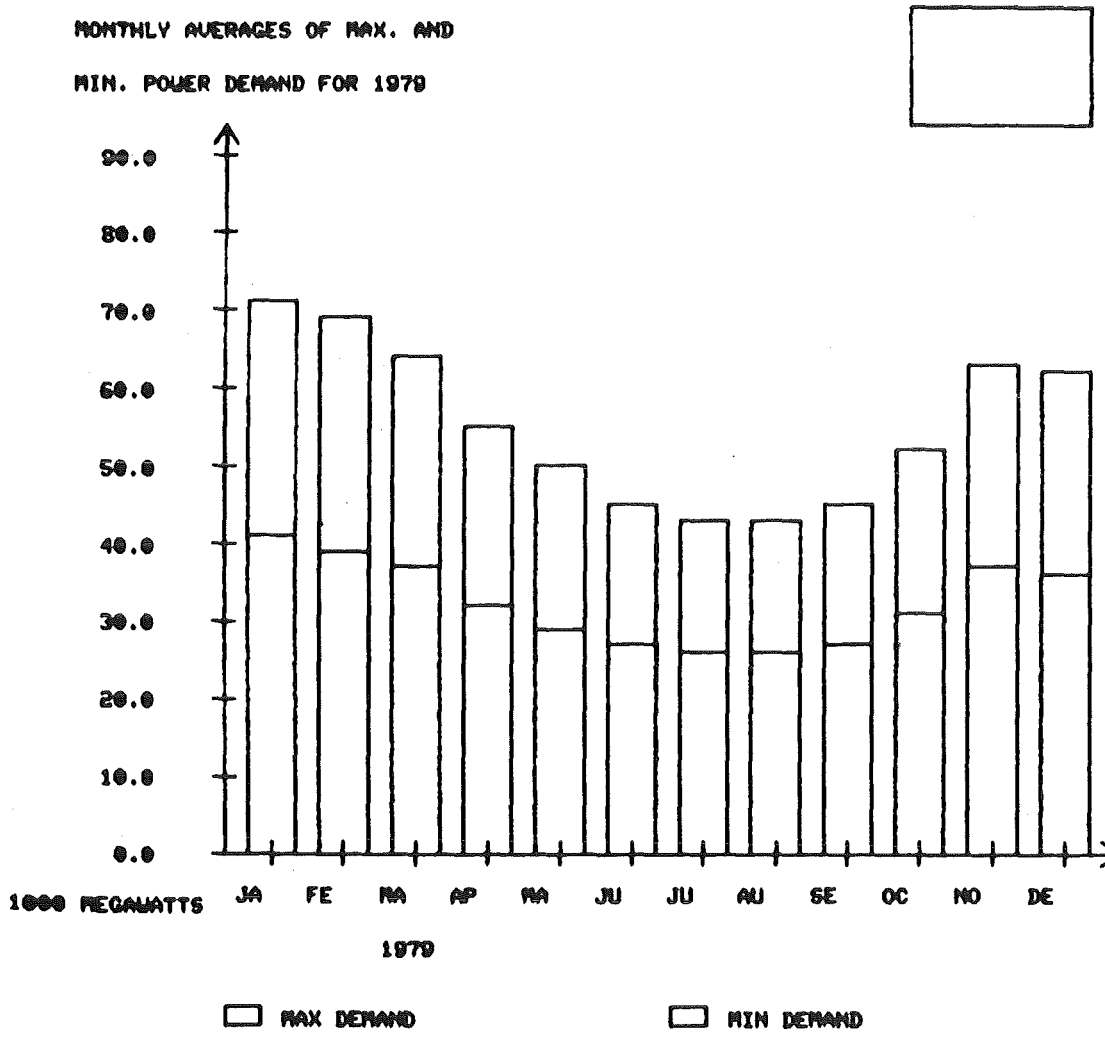


Abbildung 19: Histogramm und Rechteck

Die Farbe des Rechteckes (am AED) gefällt uns nicht. Wir wollen stattdessen die Farbe 220 (rot). Die Farbe von Flächen wird so geändert:

C: CHANGE S%0 AREA COLOUR 200 ↑
Die Farbe des Rechteckes ändert sich. S%0 ist der Name des aktuellen Teilbildes, das ist immer das zuletzt erzeugte Element, hier also das Rechteck. Wir wollen nun ein weiteres Rechteck erzeugen. Dieses soll gleich die Farbe 220 erhalten, wir ändern daher die Farbe zum Erzeugen von Rechtecken mit:

C: ATTRIBUTE AREA COLOUR 200 ↑
Wir erzeugen nun ein weiteres Rechteck:

C: RECTANGLE AREA ↑
Hier haben wir nun die Eckpunkte des Rechtecks nicht angegeben. Daher erscheint das Positionierungskreuz (AED) bzw. das Fadenkreuz (Tektronix). Wir müssen den linken unteren und den rechten oberen Eckpunkt mit Hilfe des Steuerknüppels anfahren. Ist der richtige Punkt erreicht, ist eine alphanumerische Taste zu drücken, jedoch nicht die E- oder Q-Taste. Beim Tektronix-Terminal ist nach der alphanumerischen Taste noch die ENTER-Taste zu drücken. Die Koordinaten werden auf ganze cm gerundet (wegen GRID oben) und das Rechteck gezeichnet. Diesmal erscheint es gleich in Farbe 220. Wir wollen nun in jedes der Rechtecke einen Text schreiben:

C: TEXT 'Test1' (17,18) ↑
Der Text enthält kleine Buchstaben, dafür ist am AED-Terminal vorher die Taste LOWER CASE zu drücken. Der Text erscheint in der Standardfarbe und mit Standardwerten für Höhe, Zeichensatz und den anderen Attributen. Für den nächsten Text wollen wir diese Attribute ändern und außerdem das Raster verkleinern.

C: GRID 0.5 0.5 ↑
Raster auf 1/2 cm.

C: ATTRIBUTE TEXT FONT 3 COLOUR 214 HEIGHT 1 ↑
Damit wird Zeichensatz 3, Farbe 214 und Zeichenhöhe 1 eingeschaltet.

C: TEXT ↑
Da nichts weiter angegeben ist, wird zuerst der Textstring angefordert:

TEXT:
Dieser muß nun eingegeben werden:
Test2 ↑
Nach der Eingabe des Textes erscheint das Fadenkreuz zum Positionieren des Textes. Dann wird der Text gezeichnet. Da wir nun schon einiges am Bild verändert haben, wollen wir das Zwischenergebnis auf eine SAVE-Datei retten. Die SAVE- Datei enthält stets das zuletzt gerettete Bild.

C: SAVE ↑
Wir können uns davon überzeugen, daß tatsächlich gerettet wurde, indem wir das Bild wieder einlesen:

C: RESTORE ↑
Der Bildschirm wird gelöscht und das Bild neu generiert. Nun wollen wir einige Teile des Bildes vergrößern und verschieben. Zuerst vergrößern um den Faktor 1.5:

C: SCALEP 1.5 ↑
Wir haben weder das Teilbild noch den Fixpunkt der Vergrößerung angegeben, daher meldet sich der E82 zuerst mit dem Fadenkreuz zum Aussuchen (Picken) der zu vergrößernden Bildteile. Die Teile sind anzufahren und jeweils durch Drücken einer Taste zu quittieren. Der Pickvorgang wird durch Drücken der E- oder Q-Taste abgebrochen. Dann erscheint das Fadenkreuz nochmal zum Positionieren des Fixpunktes. Alle gepickten Teile werden um den Faktor 1.5 vergrößert. Nun das Verschieben:

C: TRANSLATE ↑
Wieder wird gepickt, dann erscheint das Fadenkreuz zum Verschieben. der Verschiebevektor geht von der Stelle, an der das Kreuz erschien, zum ausgewählten Punkt. Das Bild sieht nun etwa so aus:

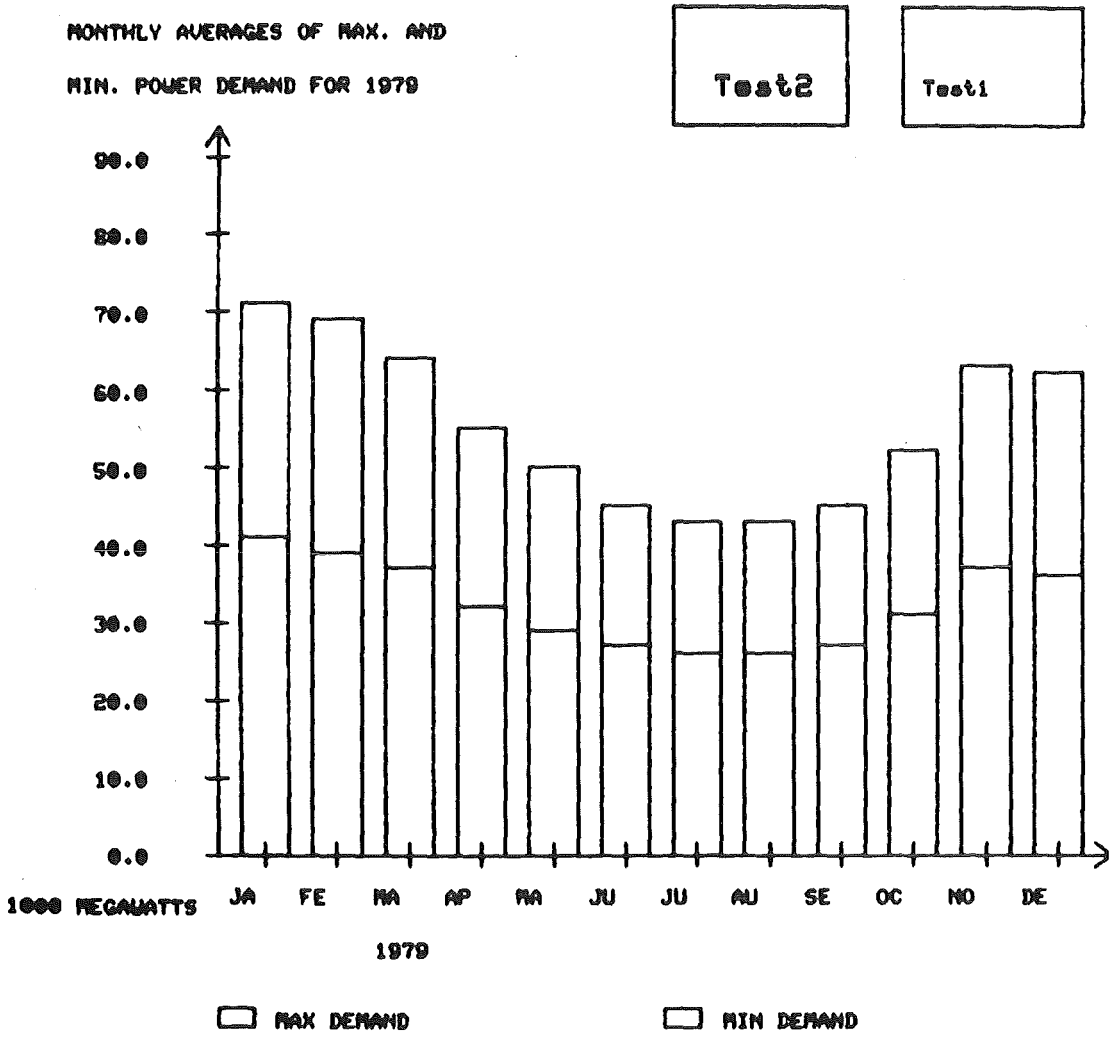


Abbildung 20: Verändertes Bild

Wir wollen nun Schluß machen:

C: END ↑
Der E82 merkt, daß wir das Bild noch nicht gerettet haben, und meldet sich mit:
CLEAR OR SAVE CURRENT PICTURE
Wir wollen das Bild auf eine Datei retten, die wir erst anlegen müssen (falls im %ALLOC ein DSN noch nicht da ist, wird die Datei neu angelegt).

C: %ALLOC NEW DSN 'TSOxxx.MYPICT.DATA' ↑
Nun können wir das Bild auf die Datei mit dem Filenamen NEW retten:

C: USE OUTPUT NEW ↑
C: WRITE 'Bild1' ↑
Durch das USE wurde unsere Datei NEW zur Ausgabedatei erklärt, mit dem WRITE wurde unser Bild auf die Datei mit dem Bildnamen Bild1 abgelegt. Nun können wir beruhigt Schluß machen:

C: END ↑
Der Editor verabschiedet sich mit dreimaligem Piepsen, dann kommen wir zurück in den TSO-Modus mit:

READY
Die Sitzung wird wie üblich abgeschlossen durch:

LOGOFF ↑.

4. BEISPIELE FÜR ANWENDUNGEN DES E82

in diesem Kapitel werden Beispiele für farbige Abbildungen gegeben, die mit Hilfe des Editors E82 erzeugt werden. Die folgende Tabelle gibt einen Überblick über die Beispiele:

Abbildung	Kommandos	Bild
21,22	PIE	Kreisdiagramme
23,24,25,26	HISTO	Histogramme
27,28	DIAGRAM	Liniendiagramme
29,30	HISTO3D	3D-Histogramme
31,32	SLIDE	Textvorlagen
33	ISO	Isoflächen
34	AREA,TEXT,CELL, POLYLINE,ATTRIBUTE	Grundelemente
35,36	SHADE	Schattierte Körper

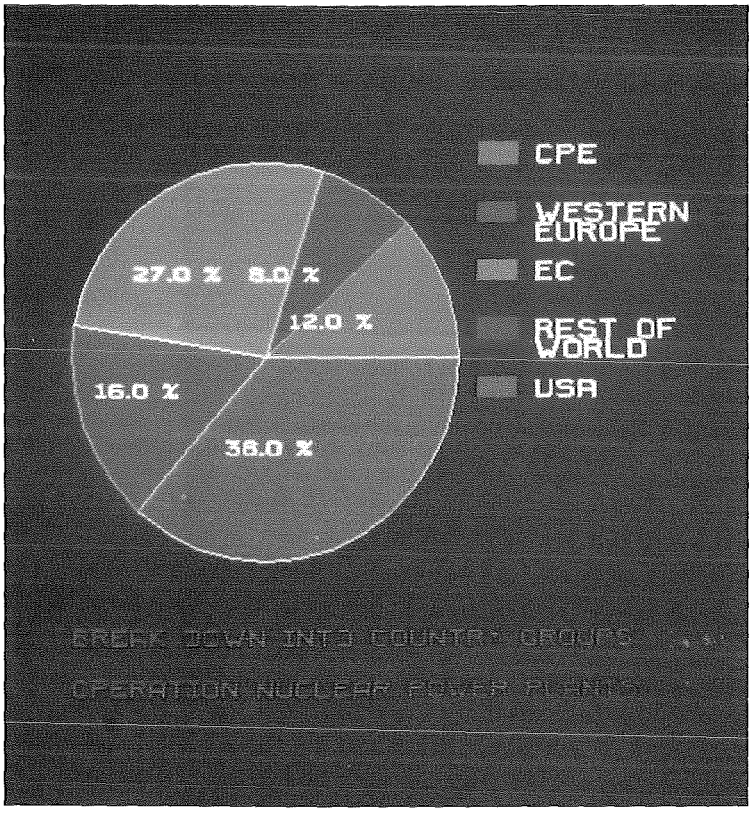


Abb.21: Kreisdiagramm

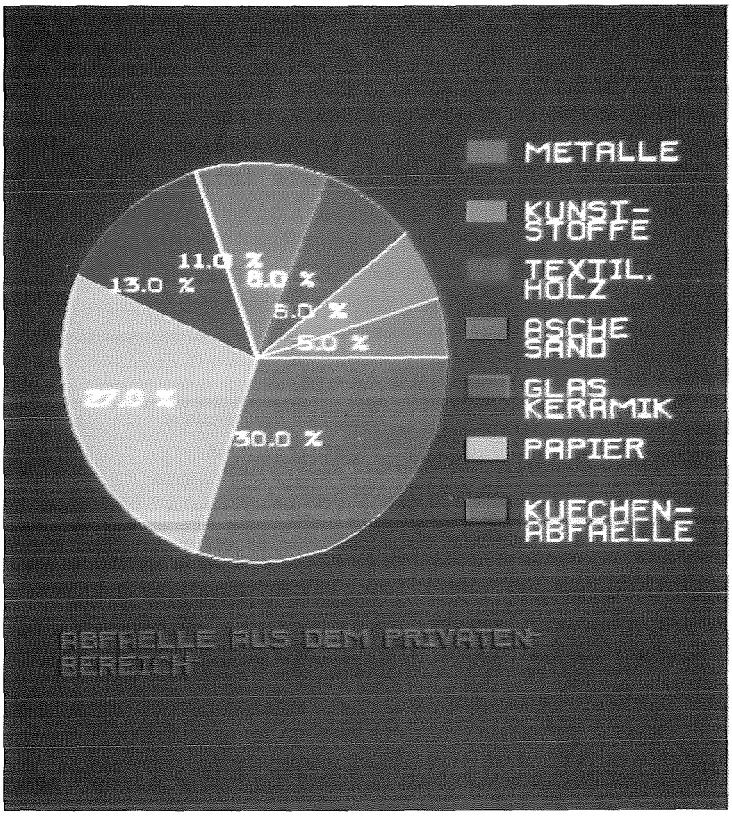


Abb.22: Kreisdiagramm

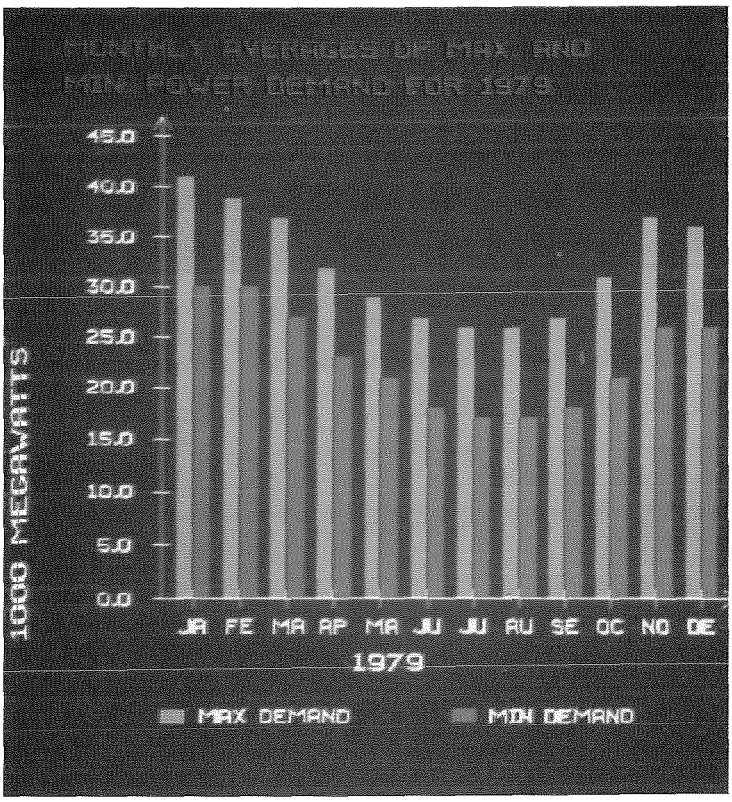


Abb.23: Histogramm

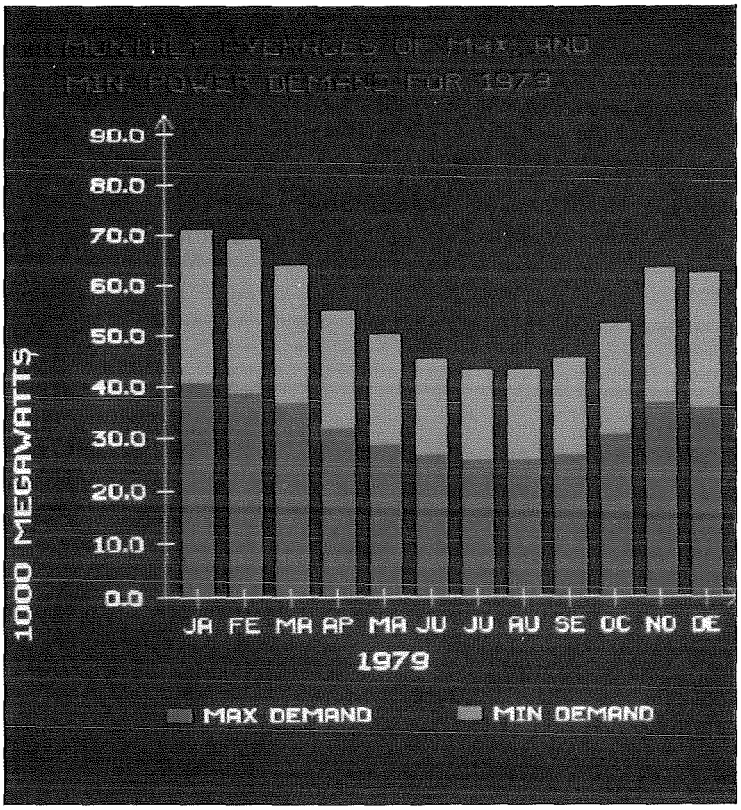


Abb.24: Histogramm

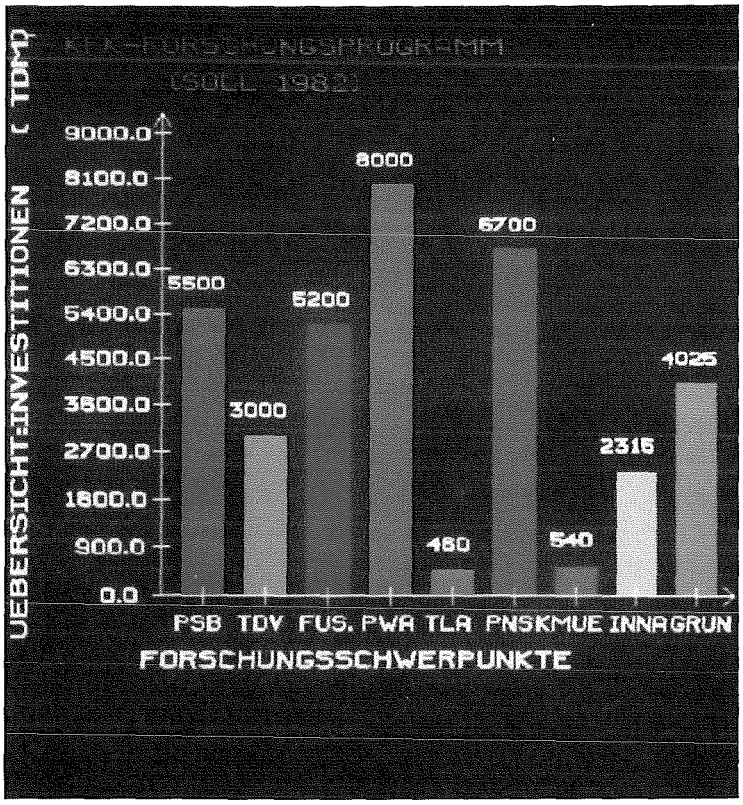


Abb.25: Histogramm

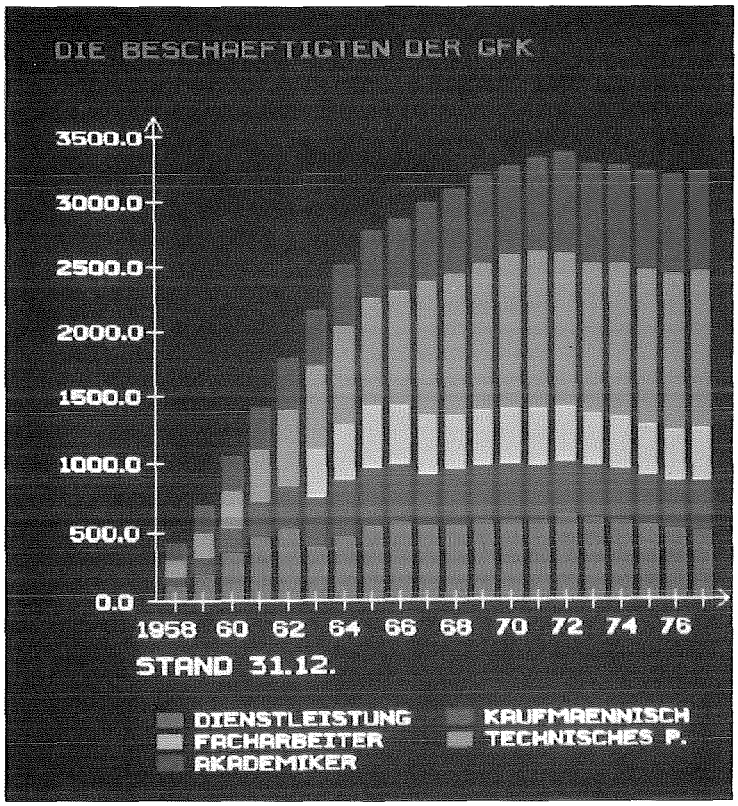


Abb.26: Histogramm

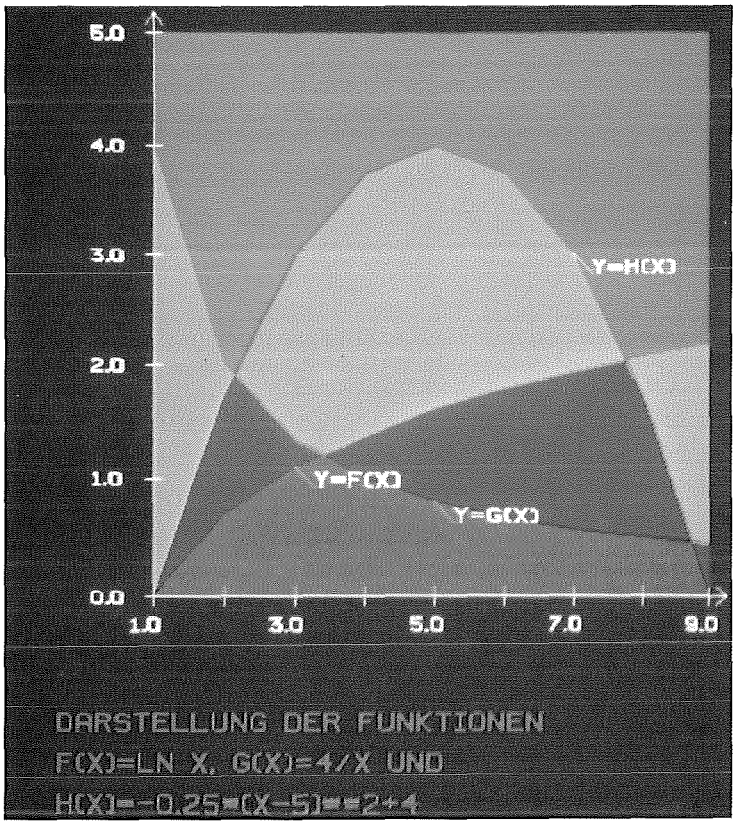


Abb.27: Liniendiagramm

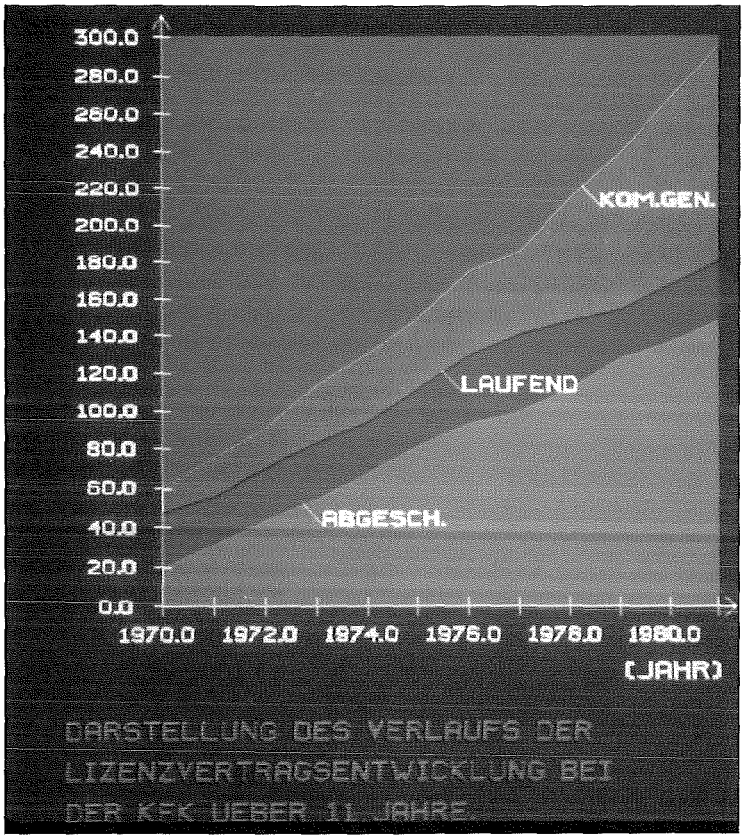


Abb.28: Liniendiagramm

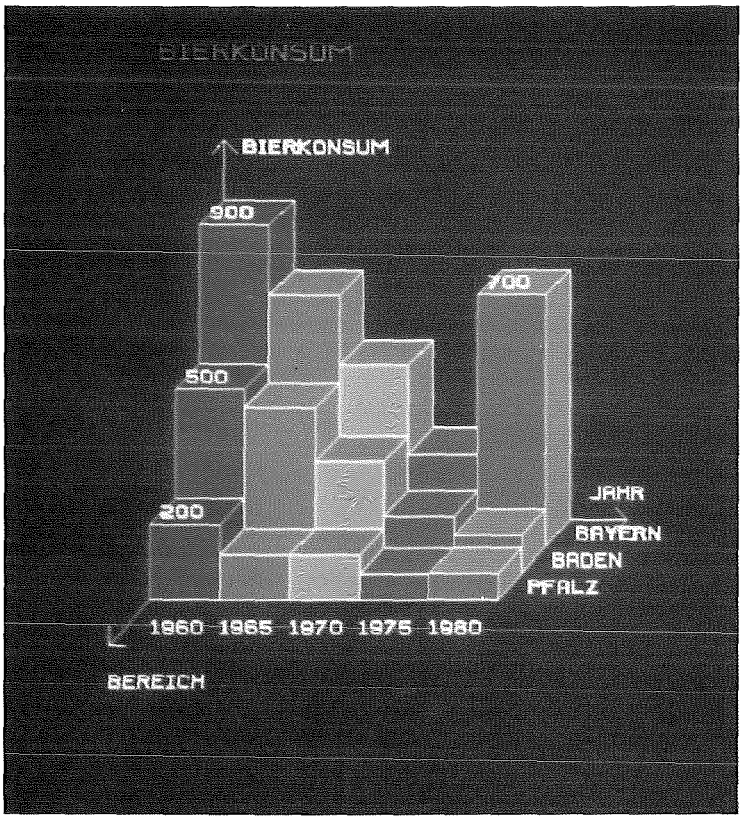


Abb.29: 3D-Histogramm

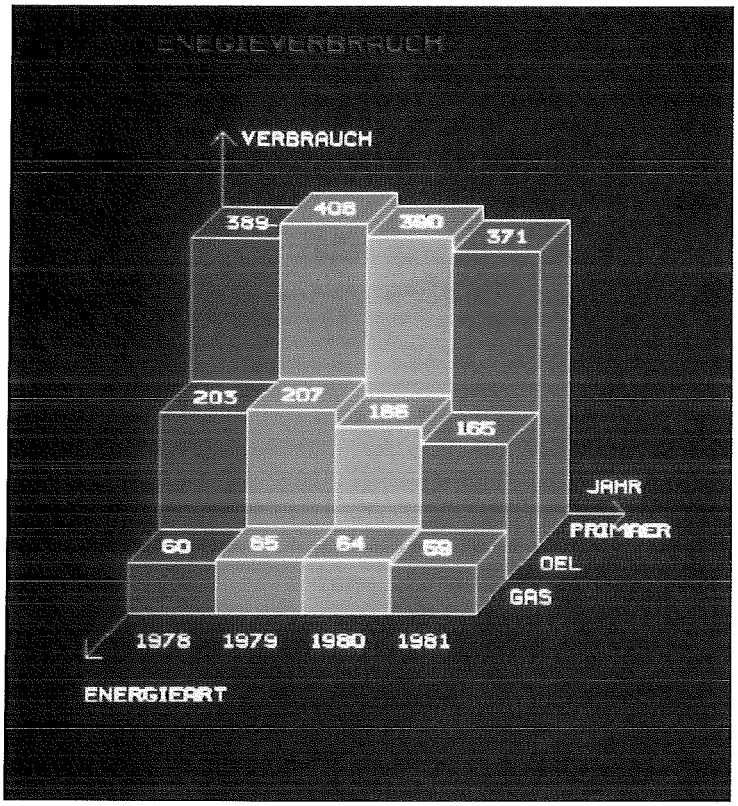


Abb.30: 3D-Histogramm

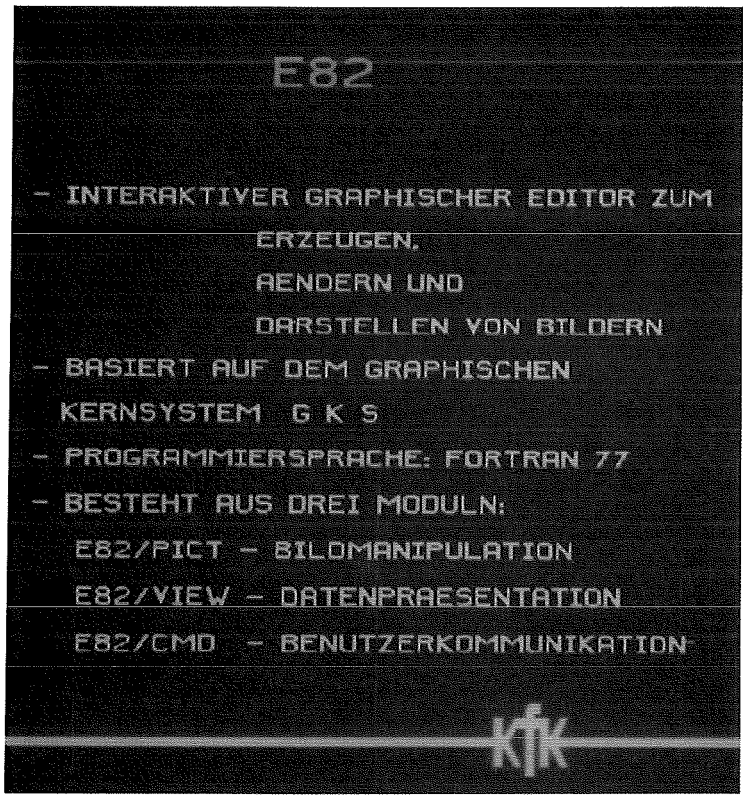


Abb.31: Textvorlage

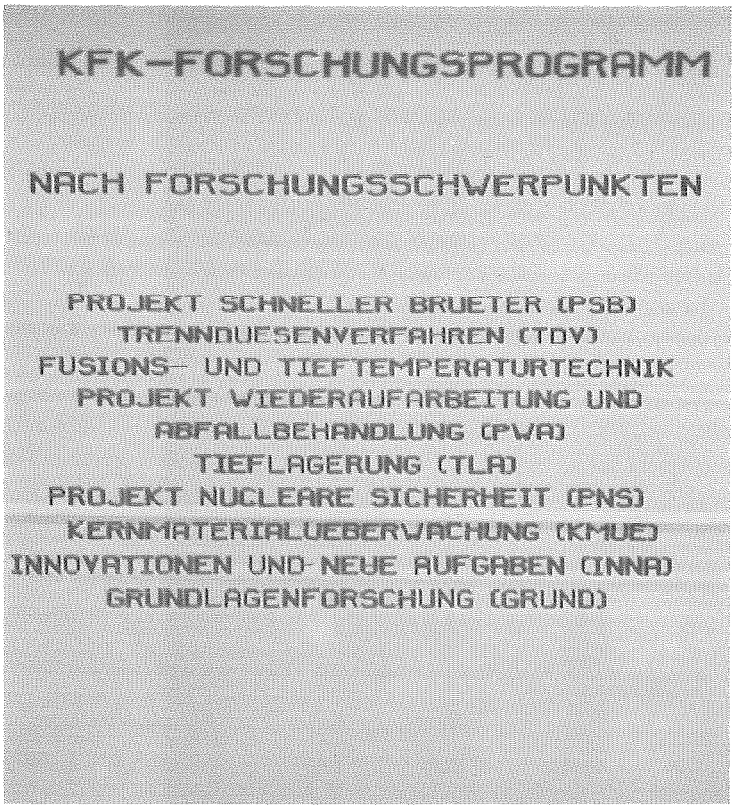


Abb.32: Textvorlage

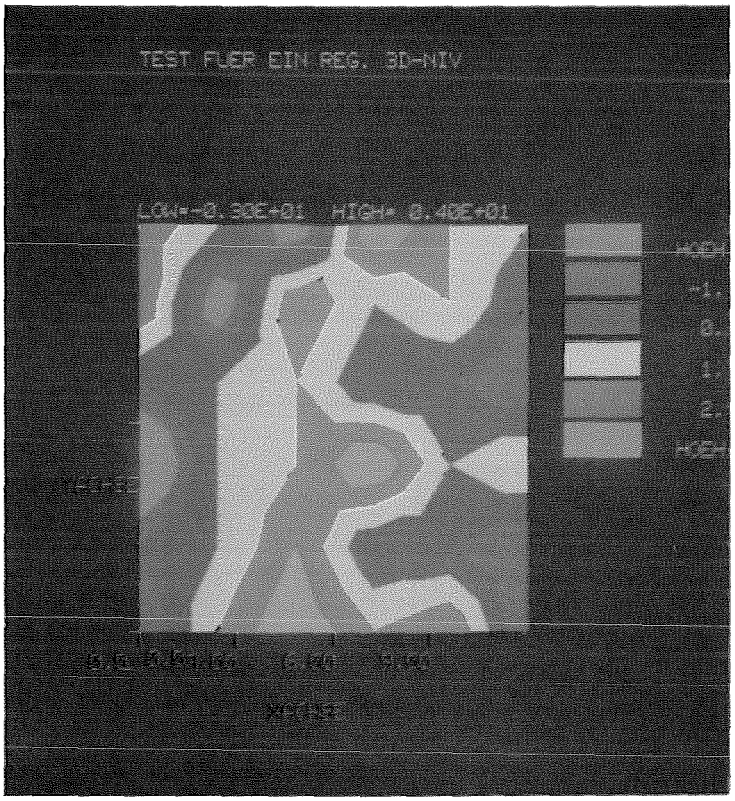


Abb.33: Isoflächen

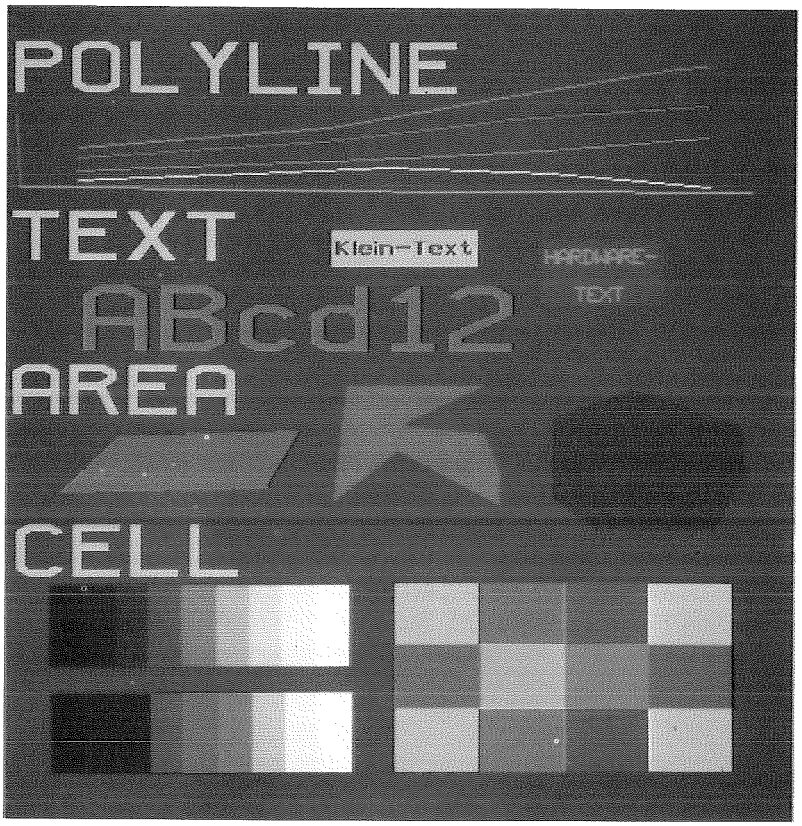


Abb.34: Grundelemente

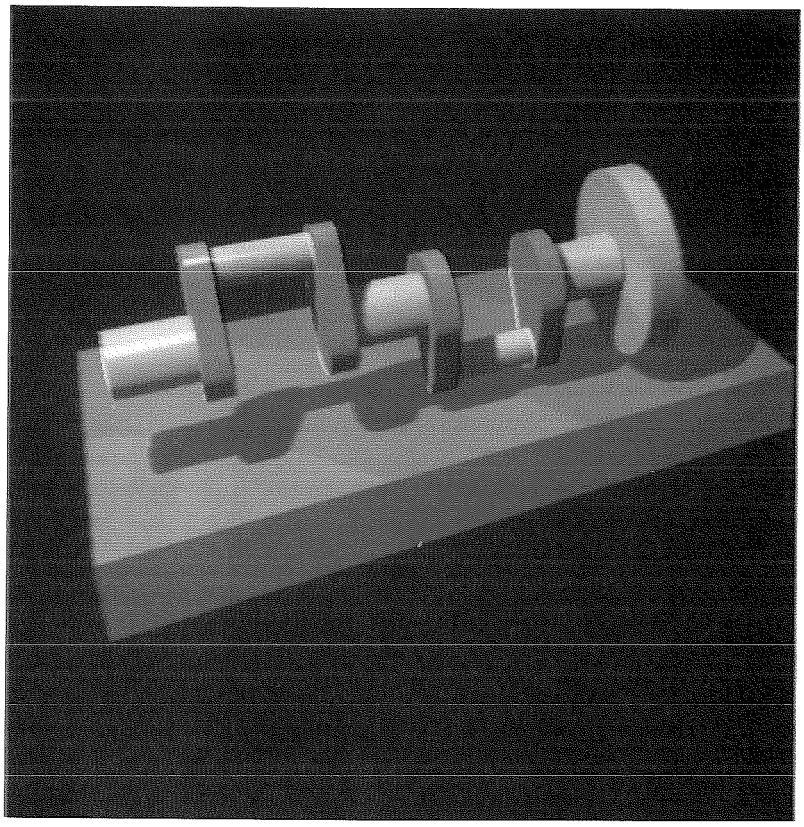


Abb.35: Schattierte Körper

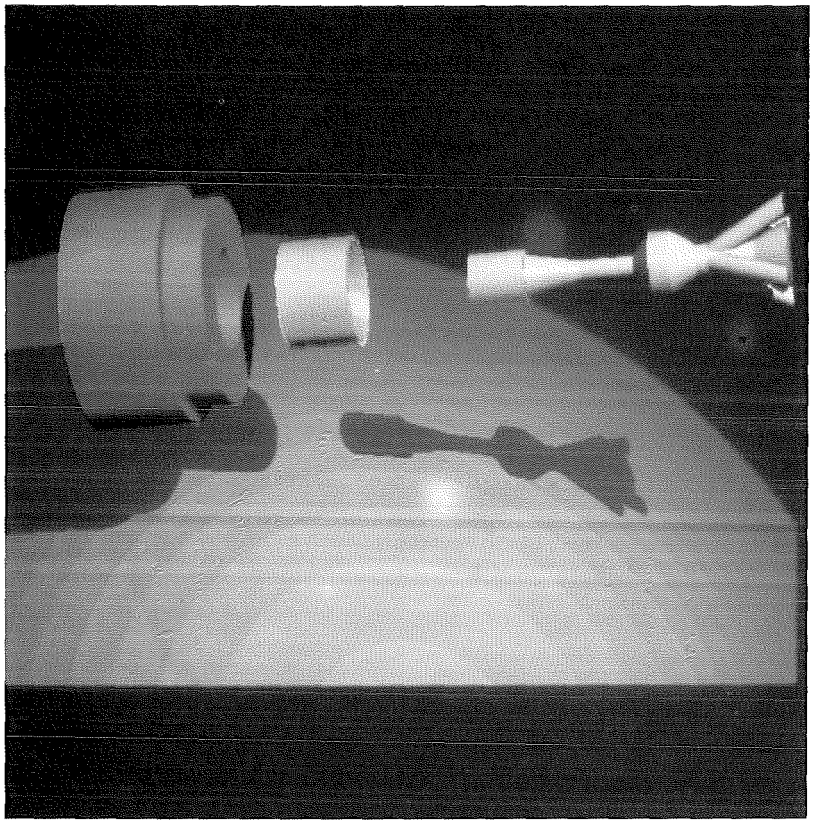


Abb.36: Schattierte Körper

5. LITERATUR

/1/ Deutsches Institut für Normung: DIN 0066252 Informationsverarbeitung, Graphisches Kernsystem, Funktionelle Beschreibung, Version 7.2, 1983

/2/ G.Enderle, I.Giese, M.Krause, H.P.Meinzer: AGF Plotfile - eine Datei zum Speichern und Transportieren Graphischer Information, Angewandte Informatik 21, Nr.6, 1979

/3/ G.Enderle, K.H.Bechler, F.Katz, K.Leinemann, W.Olbrich, E.G. Schlechtendahl, K. Stölting: GIPSY-Handbuch, KfK 2878 und KfK-CAD146, 1979

/4/ W.Zimmerer: PLOTCP, ein Fortran-Programm zur Erzeugung von Calcomp-Plot-Zeichnungen, KfK 2081, Mai 1975

/5/ G.Enderle: Definition, Übersetzung und Anwendung benutzerorientierter Sprachen als Erweiterung von PL/1 in dem System für das rechnergestützte Entwerfen und Konstruieren REGENT, PLS, KfK 2204, 1975

/6/ Deutsches Institut für Normung: FORTRAN-Binding for GKS 7.2, DIN-NI/UA-5.9/42-82, 1982.

/7/ G.Enderle, K.H.Bechler, H.Grimme, W.Hieber, F.Katz: GIPSY-Handbuch Band II, KfK 3216, 1982