KERNFORSCHUNGSZENTRUM

KARLSRUHE

Dezember 1976 KFK 2415

Institut für Datenverarbeitung in der Technik

Einführung in die graphische Programmierung mit LLGL

W. Müller, H. Grauer

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

KFK 2415

Institut für Datenverarbeitung in der Technik

Einführung in die graphische Programmierung mit LLGL

W. Müller
H. Grauer

Gesellschaft für Kernforschung m.b.H., Karlsruhe

Kurzfassung

Zur komfortablen und effizienten Entwicklung von interaktiven graphischen Programmen ist eine benutzerfreundliche und leistungsfähige interaktive graphische Sprache eine wesentliche Voraussetzung.
Dieser Bericht soll eine Anleitung zur Benutzung einer solchen Sprache geben und gleichzeitig als Tutorial für Anwender dienen, die sich in die Problematik von leistungsfähigen interaktiven graphischen Sprachen anhand eines konkreten Beispiels einarbeiten möchten.
Es werden die wesentlichen Eigenschaften der interaktiven graphischen Sprache LLGL (Low-Level-Graphical-Language) dargestellt und eine ausführliche Beschreibung der einzelnen LLGL-Befehle, verbunden mit Programmierbeispielen, gegeben. Abschließend wird die Anwendung von LLGL anhand der Programmierung eines Systems zum interaktiven Entwurf von Schaltnetzen gezeigt.

Introduction into graphical programming with LLGL

Abstract

For comfortable and productive development of interactive graphical programs a user-oriented and efficient interactive graphical language is an essential prerequisite.
This paper shall be a guide for the use of such a language and at the same time a tutorial for those users, who want to learn on a concrete example about the problems of efficient interactive graphical languages.
The essential features of the interactive graphical language LLGL (Low-Level-Graphical-Language) are shown and a detailed description of the LLGL instruction set is given, combined with programming examples. Finally the application of LLGL is demonstrated by programming a system for interactive development of logic circuits.

Inhalt

Einleitung

LLGL (Low Level Graphical Language) ist eine interaktive graphische Sprache zur komfortablen Programmierung von Problemen, die eine intensive graphische Mensch-Maschine-Kommunikation erfordern.

LLGL ist anwendungsinvariant und verfügt über leistungsfähige Sprachelemente zum Aufbau und zur interaktiven Änderung von hierarchisch strukturierten Bildern. Die Sprache ist als von FORTRAN aus benutzbares Unterprogrammpaket implementiert.

LLGL ist wohlstrukturiert aufgebaut und die einzelnen Befehle sind übersichtlich parametrisiert, so daß die graphische Programmierung mit LLGL schnell erlernbar ist.

Dieser Bericht ist so aufgebaut, daß er nicht nur ein Benutzerhandbuch für die Programmierung mit LLGL darstellt, sondern darüber hinaus dem Anwender, der sich in den Problemkreis der interaktiven graphischen Programmierung anhand eines konkreten Beispiels einarbeiten möchte, als Tutorial dienen kann.

Es wird zunächst ein kurzer Überblick über die wesentlichen Eigenschaften von LLGL gegeben (eine detaillierte Darstellung wird in /1,2,3,4,5/ gegeben). Danach folgt eine ausführliche Beschreibung der einzelnen Befehle zusammen mit Programmierbeispielen. Abschließend wird die umfassende Anwendung von LLGL anhand der Programmierung eines Systems zum interaktiven Entwurf von Schaltnetzen gezeigt.

1. Konfiguration

LLGL ist zur Zeit auf einem Varian V74 Kleinrechner mit ADAGE GP410 Bildschirmgerät implementiert.

1.1 Physikalische Geräteeinheit

Der Kern des Bildschirmgerätes ist ein schneller, mikroprogrammierbarer Mikroprozessor, der Displayprozessor. Er interpretiert den im Rechner liegenden Displayfile, in dem die Ausgabeinformation abgelegt ist. Diese Daten werden geeigneten Manipulationen (z.B. geometrischen Transformationen) unterworfen und auf eine Elektronenstrahlröhre nach dem random-scan-Verfahren ausgegeben. Die Interpretation des Displayfile wird zyklisch alle ca. 20 ms wiederholt, sodaß für den Betrachter der Eindruck eines stehenden Bildes entsteht.

Der Displayprozessor überwacht parallel zur Bildausgabe die Eingabegeräte (Tastaturen, Lichtgriffel, ...). Er übernimmt die von ihnen generierten Eingabeinformationen und signalisiert eine erfolgte Eingabe über einen Interrupt an den Rechner.

1.2 Logische Einheiten

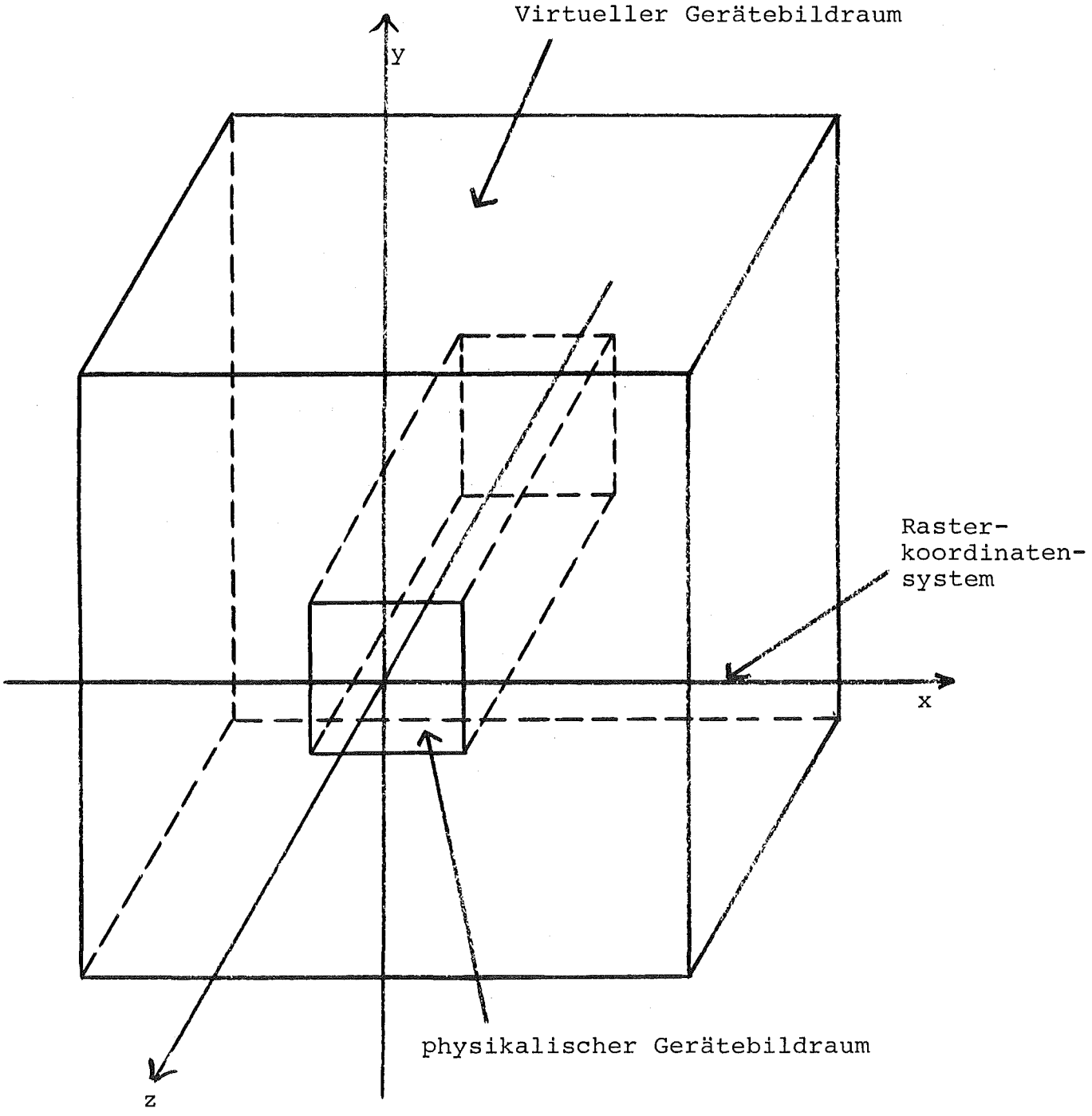
LLGL verdeckt diese Hardwaredetails. Für einen Benutzer existieren logische Eingabeeinheiten und eine logische Ausgabeeinheit.

1.2.1 Logische Ausgabeeinheit

Die logische Ausgabeeinheit stellt sich als ein Bildraum (im Gegensatz zur zweidimensionalen Bildfläche eines Plotters) in Form eines Würfels mit einer Kantenlänge von etwa 1.2 m (genau 48 inch) dar. Dieser Bildraum wird im folgenden als virtueller Gerätebildraum bezeichnet. In der Mitte dieses Raumes

liegt der Ursprung eines rechthändigen, rechtwinklig cartesischen Koordinatensystems, des Rasterkoordinatensystems.
Auf jeder Halbachse dieses Systems sind maximal 32 768 Rasterpunkte adressierbar (Die z-Koordinate wird durch Helligkeitsmodulation simuliert; größerer z-Wert entspricht einer größeren Helligkeit). Bedingt durch die Auflösung der Hardwareeinheiten werden in x- und y-Richtung jeweils acht, in z-Richtung jeweils 1024 aufeinander folgende Adressen auf den gleichen Rasterpunkt abgebildet. Daraus errechnet sich der minimale Abstand zweier Rasterpunkte in der x-y-Ebene zu etwa 0.15 mm, während man in z-Richtung 64 verschiedene Werte (=Helligkeit) unterscheiden kann. In dem so quantisierten Raum werden Linien mit einer Strichstärke von etwa 0.5 mm gezeichnet.
Koordinatenbezogene Daten werden vom Benutzer mit Bezug auf das Rasterkoordinatensystem als Realzahlen angegeben. Dabei kann er über das Attribut Einheiten mit dem zugehörigen Befehl die Einheitenlängen für jede der drei Achsen festlegen, wobei auch negative Längen (bedeutet eine Änderung der Achsenrichtung) zugelassen sind.

Aus dem virtuellen Bildraum werden quaderförmige Ausschnitte auf den Darstellungsraum der Bildröhre, den physikalischen Gerätebildraum, abgebildet. Die positive x-Halbachse ist dabei nach rechts, die positive y-Halbachse nach oben und die positive z-Halbachse nach vorn, auf den Betrachter hin, gerichtet. Der physikalische Bildraum hat eine Ausdehnung von etwa 0.30 m (12 inch) in x- und y-Richtung und von etwa 1.2 m (48 inch) in z-Richtung.
Die Ausdehnung des Ausschnittquaders in x- und y-Richtung kann vom Benutzer über das Attribut Ausschnitt mit dem zugehörigen Befehl gesetzt werden. LLGL skaliert dann den Ausschnitt automatisch so, daß er den physikalischen Bildraum voll ausfüllt.


Virtueller Gerätebildraum
y
Raster-
koordinaten-
system
x
physikalischer Gerätebildraum
z

Transformationen in der logischen Ausgabeeinheit

Alle in dem Displayfile enthaltenen Koordinaten, das sind Start- und Endpunkte von Vektoren, Stützpunkte von Polygonen und Anfangspunkte von Texten, werden vor ihrer Ausgabe geometrischen Transformationen unterworfen. Diese Abbildungen werden beschrieben durch

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} S \end{pmatrix} \begin{pmatrix} R \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} \qquad (1)$$

S	Skalierungsmatrix. Setzt einen Verkleinerungsfaktor für jede der drei Koordinatenachsen.
R	Rotationsmatrix. Bestimmt eine Drehung um einen für jede Achse angebbaren Winkel.
dx,dy,dz	Translationsvektor. Legt eine Translation in x-, y- und z-Richtung fest.

Texte werden folgenden Transformationen unterworfen:

1. Skalierung
   Die Zeichengröße, der Zeichen- und der Linienabstand können mit einem Verkleinerungsfaktor multipliziert werden (Maximale Zeichenhöhe ist ca. 33 mm bei einem Höhen/Breitenverhältnis der Großbuchstaben von 1,4:1)
2. Rotation
   - Einzelne Zeichen lassen sich um ein ganzzahliges Vielfaches von 90 Grad drehen. Die Drehachse verläuft parallel zur z-Achse
   - Die Schreibrichtung innerhalb einer Zeile kann in vier verschiedene Richtungen zeigen
   - Entsprechend kann auch die Anordnung der Zeilen gesteuert werden

3. Spiegelung
   Einzelne Zeichen können zusätzlich zur Rotation auch um ihre vertikale Achse gespiegelt werden

Diese Transformationen, Koordinaten und Zeichen betreffend, werden von Hardwareeinheiten durchgeführt. In der aktuell vorhandenen Geräteversion ADAGE A410 sind die Zeichentransformationen vollständig realisiert. Die Fähigkeiten zur Koordinatentransformation sind dagegen beschränkt auf:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} scale & & \\ & scale & \\ & & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} dx \\ dy \\ 0 \end{pmatrix}$$

d.h.

$$S = scale \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ und } R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

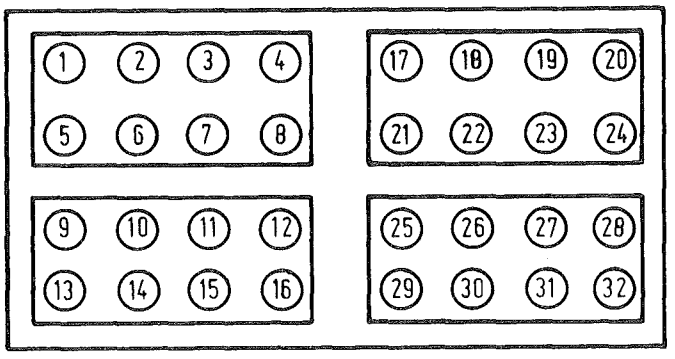
1.2.2 Logische Eingabeeinheiten

Die physikalischen Eingabeeinheiten des Sichtgerätes sind für den Benutzer von LLGL logisch beschrieben durch die Funktion(en), die ihnen in LLGL zugewiesen werden, die daraus folgende Vorschrift für ihre Bedienung, sowie durch die Art und das Format der Daten, die bei Eintreten eines Eingabeereignisses an das Benutzerprogramm übergeben werden.

Es gibt folgende Eingabeeinheiten:

Funktionstastatur

Die Funktionstastatur dient bevorzugt zur Eingabe von Steuerinformation zur Steuerung des Programmkontrollflusses.
Sie besteht aus einem Tastenfeld mit 32 Tasten in folgender Anordnung:



Jede Taste entspricht einem bestimmten Zustand der Eingabeeinheit, der durch eine ganze Zahl i, $1 \leq i \leq 32$, gekennzeichnet ist. Nach der Initialisierung befindet sich die Eingabeeinheit in einem undefinierten Zustand.

Durch Betätigen einer Taste tritt ein Zustandswechsel ein. Die dem neuen Zustand zugeordnete ganze Zahl wird dabei an das Benutzerprogramm übergeben (Zustandswechsel = Eingabeereignis). Die Einnahme eines Zustandes wird durch Erleuchten der betreffenden Taste gekennzeichnet.

Alphanumerische Tastatur

Mit der alphanumerischen Tastatur wird Zeicheninformation an das Benutzerprogramm übergeben.
Der Zeichenvorrat der alphanumerischen Tastatur entspricht dem in Tabelle 2 (siehe Seite 55). Beim Drücken einer Taste wird ein Eingabeereignis ausgelöst und der ASCII-Code der betreffenden Taste rechtbündig angeordnet an das Benutzerprogramm übergeben.

Lichtgriffel

Der Lichtgriffel ist die Eingabeeinheit zur Durchführung der Zeigefunktion (=Deuten auf Linien und Zeichen auf dem Bildschirm). Mit ihm können Ausgabeobjekte auf dem Bildschirm

identifiziert werden.
Er wird durch Niederdrücken eines Schalters am Lichtgriffel aktiviert.
Bei Eintreten eines Eingabeereignisses wird der Wert des Menüzählers und die Namenshierarchie des auslösenden Ausgabeobjektes (siehe Tabelle Seite 20) an das Benutzerprogramm übergeben.

Der Lichtgriffel kann in zwei Betriebsarten benutzt werden:

a) Modus 'Pick'

Nach der Aktivierung des Lichtgriffels kann durch Zeigen auf ein Ausgabeobjekt genau ein Eingabeereignis ausgelöst werden. Das nächste Eingabeereignis kann erst nach Deaktivierung und erneuter Aktivierung durch Loslassen und neues Niederdrücken des Lichtgriffelschalters ausgelöst werden. Der Modus 'Pick' ist der normale Modus zur Durchführung der Zeigefunktion. Mit ihm wird das Auftreten unerwünschter Eingabeereignisse verhindert.

b) Modus 'Track'

Nach der Aktivierung des Lichtgriffels können durch Zeigen auf ein Ausgabeobjekt beliebig viele Eingabeereignisse ausgelöst werden. Der Modus 'Track' kann zur Realisierung einer Tracking-Funktion (Nachführen eines Tracking-Symbols mit dem Lichtgriffel auf dem Bildschirm) benutzt werden.

Programmierte Eingabe

Die programmierte Eingabe nimmt als Eingabeeinheit eine Sonderstellung ein. Sie wird durch ein Ausgabeobjekt 'Programmierte Eingabe' im Display-File bei jedem Durchlauf des Displayprozessors durch das Display-File erzeugt.
Bei einem Eingabeereignis wird der Wert des Ausgabeobjektes 'Programmierte Eingabe' (vom Benutzer angegebene ganze Zahl i, $32767 \leq i \leq 32767$) sowie die aktuelle Strahlposition an das Benutzerprogramm übergeben.

2. Eigenschaften von LLGL

2.1 Allgemeines

LLGL ist eine interaktive graphische Sprache, welche eine benutzerfreundliche und effektive Programmierung interaktiver graphischer Problemstellungen ermöglicht. Sie zeichnet sich durch folgende Eigenschaften aus:

a) Strukturierung von Bildern

LLGL ermöglicht den Aufbau von strukturierten Bildern. Die Strukturierung eines Bildes in graphische Objekte soll zum einen die Definition von Bildern erleichtern, indem mehrfach auftretende graphische Objekte nur einmal erstellt werden müssen und anschließend an beliebigen Stellen im Bild benützt werden können (graphische Unterprogrammtechnik) und zum anderen die bequeme Manipulation von graphischen Objekten durch Attributänderung gestatten (z.B. Änderung der Textur von Bildteilen, Durchführung von Transformationen). Dazu müssen graphische Objekte hierarchisch zusammengefaßt werden können, um Attribute bzw. Attributänderungen gleichzeitig auf eine gewünschte Menge von Objekten wirksam werden zu lassen (graphische Blockstruktur). Die Struktur eines Bildes ist nicht fest, sondern kann jederzeit durch Löschen von vorhandenen und Einfügen von neuen graphischen Objekten geändert werden (wichtig für interaktiven Betrieb).

b) Identifizierung von graphischen Objekten

Der Benutzer kann graphische Objekte identifizieren. Die Identifizierung erfolgt im Programm über einen dem graphischen Objekt dort explizit zugewiesenen Namen oder interaktiv durch direktes Zeigen mit dem Lichtgriffel auf das Objekt. Das Ergebnis der Zeigeoperation ist der im Programm zugewiesene Name oder ein Pseudoname, falls das Objekt noch keinen Namen hatte, der dann wie der explizit zugewiesene Name benutzt werden kann.

c) Unabhängigkeit vom physikalischen Koordinatensystem des Bildschirms.
Der Benutzer kann in einem eigenen nur durch die Genauigkeit des Rechners begrenzten reellen Koordinatensystem arbeiten, welches er durch Angabe einer Abbildungsfunktion geeignet auf den Bildschirm abbilden kann.

d) Flexible und leistungsfähige Verarbeitung von Eingaben.
Zur Verarbeitung von Eingaben können diesen beliebige Programme oder Tasks zur Laufzeit zugeordnet werden.
Die Verarbeitung kann synchron durch Abfragen (Polling) oder durch Warten auf eine Eingabe erfolgen. Durch die Zuordnung von Eingaben zu eigenen Tasks ist außerdem eine asynchrone prioritätsgestaffelte Verarbeitung möglich.

e) Benutzerfreundliche Sprachdefinition

LLGL stellt sich dem Benutzer als eine graphische Spracherweiterung von FORTRAN dar. Die Sprachoperatoren haben die Form von Unterprogrammaufrufen. Die Anzahl der zu einem Operator gehörenden Operanden (Parameter der Unterprogrammaufrufe) wurde bewußt klein gehalten, um eine übersichtliche Programmierung zu ermöglichen.

2.2 Objektarten der Sprache LLGL

Primitive Objekte

LLGL kennt die folgenden vier primitiven Objekte:

- Polygon: In dem virtuellen Bildraum wird mit diesem Objekt ein Linienzug erzeugt, der eine beliebige Anzahl von Stützpunkten verbindet.
- Vektor: Dieses Objekt erzeugt im virtuellen Bildraum einen Vektor.
- Symbolstring: Mit diesem Objekt wird an einem beliebigen Ort im virtuellen Bildraum eine Zeichenkette ausgegeben. Es steht

dabei ein Zeichenvorrat von 128 Zeichen (95 ASCII-Zeichen + Sonderzeichen) zur Verfügung. Mit diesem Objekt kann man außerdem Menüs aufbauen.

- Referenz: Dieses Objekt referenziert ein höheres referenzierbares Objekt. Das höhere Objekt wird dadurch sichtbar gemacht. Das Referenzobjekt ist einem Unterprogrammaufruf in einer Programmiersprache vergleichbar.
- Programmierte Eingabe: Dieses Objekt erzeugt ein Eingabeereignis von dem gleichnamigen Eingabegerät. Dieses Ereignis wird bei jedem Durchlauf des Displayprozessors durch den Displayfile, i.a. also alle ca. 20 ms, erzeugt.

Höhere Objekte

Primitive Objekte können zusammengefaßt werden zu höheren Objekten. Man unterscheidet bei dieser Objektart zwischen dem referenzierbaren und dem nicht referenzierbaren, höheren Objekt.
Das nicht referenzierbare, höhere Objekt dient in erster Linie der Zusammenfassung mehrerer Objekte zum Zwecke der leichten Manipulation (so kann man z.B. die Textur aller in einem solchen Objekt enthaltenen Teilobjekt gleichzeitig setzen, indem nur die Textur des zusammenfassenden Objekts geändert wird).
Das referenzierbare, höhere Objekt hat darüberhinaus noch eine Funktion, die mit dem Unterprogramm in einer Programmiersprache vergleichbar ist. Ein solches Objekt wird einmal definiert und kann dann an mehreren Stellen durch das Referenzobjekt aufgerufen, d.h. sichtbar gemacht werden.

Parameterliste

Für die primitiven Objekte, Vektor, Polygon und Symbolstring können Parameterlisten (Felder vom Typ Vektor, Polygon bzw.Symbolstring) definiert werden, die ähnlich wie die höheren referenzierbaren Objekte an mehreren Orten 'aufgerufen' werden

können.

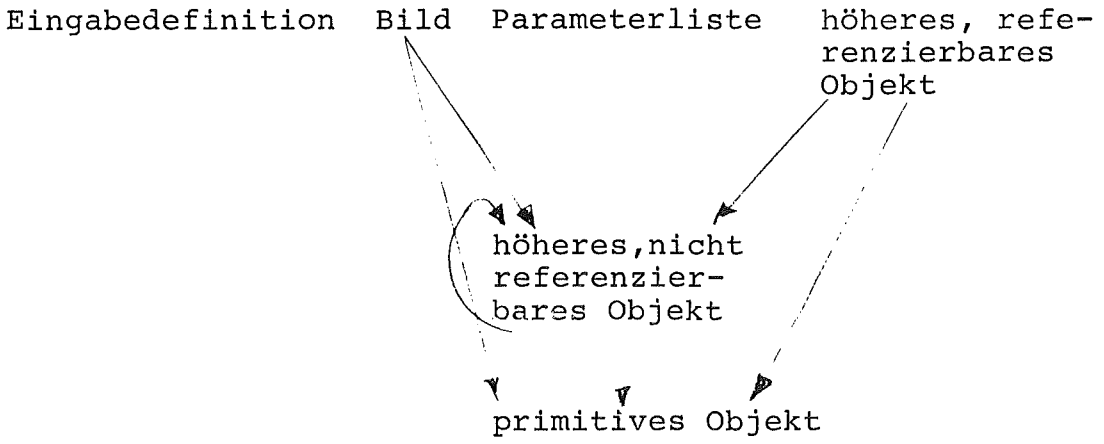
Bild

Jeder Benutzertask ist genau ein Objekt Bild statisch zugeordnet. In ihm sind alle von dieser Task generierten Objekte, mit Ausnahme von Parameterlisten, Eingabedefinitionen und höheren, referenzierbaren Objekten, enthalten.

Eingabedefinitionen

Mit diesem Objekt wird eine Zuordnung zwischen einem Eingabegerät und einer Task hergestellt. Außerdem wird ein Datenbereich zur Verfügung gestellt, in dem die Eingabeinformation abgelegt wird.

Die Objekte haben somit die folgende Struktur:



--▸ := besteht aus

Den oben beschriebenen Objekten sind eine Menge von Attributen zugeordnet. Diese dienen u.a. zur Selektion und Modifizierung von Objekten. Die Attribute und ihre Bedeutung sind:

Ausschnitt	Bestimmt die Größe des Ausschnitts des virtuellen Gerätebildraums, der auf dem physikalischen Gerätebildraum abgebildet wird

Einheit	Bestimmt die Einheitenlänge auf den Koordinatenachsen des Rasterkoordinatensystems
Sichtbar	Bestimmt, ob das einer Task zugeordnete Bild sichtbar ist oder nicht
Lichtgriffel	Bestimmt, ob ein Objekt vom Lichtgriffel gesehen werden kann oder nicht
Blinken	Bestimmt, ob ein Objekt blinkt oder nicht
Name	Ordnet einem Objekt einen Benutzernamen zu
Translation	Bestimmt den Translationsvektor, um den ein Objekt verschoben wird
Vektorskalierung	Bestimmt einen Verkleinerungsfaktor für ein Objekt
Vektortextur	Bestimmt die Textur in der Linien ausgegeben werden
Symbolskalierung	Bestimmt einen Verkleinerungsfaktor für Texte
Symboltextur	Bestimmt die Textur, in der Texte ausgegeben werden
Symboldrehung	Bestimmt einen Drehwinkel für Einzel-Zeichen
Schreibrichtung	Bestimmt die Schreibrichtung, in der Texte ausgegeben werden
Unterstreichung	Bestimmt, ob die Unterstreichungszeichen in einem Text blinken oder nicht

Über die möglich Zuordnung von Attributen zu Objekten gibt die Tabelle 1 Auskunft.

Objektart / Attribut	Bild	höher	Parameter-liste	Vektor/ Polygon	Symbol-string	Referenz	Programmierte Eingabe	Eingabedefi-nition
Name	-	x	x	x	x	x	x	-
Lichtgriffel	sichtbar	x	u.	x	x	x	-	-
Blinken	blinke nicht	x	u.	x	x	x	-	-
Vektortextur	durchgezogene	x	u.	x	-	x	-	-
Symboltextur	Linien	x	u.	-	x	x	-	-
Translation	keine	x	u.	u.	u.	x	-	-
Vektorskalierung	keine	x	u.	u.	-	x	-	-
Symbolskalierung	keine	x	u.	-	u.	x	-	-
Schreibrichtung	übliche Konvention	x	u.	-	u.	x	-	-
Symboldrehung	keine	x	u.	-	u.	x	-	-
Unterstreichung	blinke nicht	x	-	-	x	-	-	-
Einheiten	x	-	-	-	-	-	-	-
Ausschnitt	x	-	-	-	-	-	-	-
Sichtbar	x	-	-	-	-	-	-	-

x : Attribut ist zulässig, Wert beliebig aus dem Wertevorrat

Text: Attribut ist zulässig, Wert ist konstant (=Text) und kann nicht geändert werden [u. = umgebungsabhängig*)]

- : Attribut ist nicht zulässig

Tabelle 1 Zuordnung der Attribute zu den Objekten

Diese Attribute sind von LLGL mit änderbaren Defaultwerten vorbelegt. Diese Defaultwerte sind wie folgt festgelegt:

Ausschnitt: -8192, -8192, 8192
Einheiten : 1, 1, 1
Name : Pseudonamen

*) Der Wert 'umgebungsabhängig' bedeutet, daß der Wert dieses Attributs aus dem hierarchisch darüberliegenden Objekt genommen wird

Sichtbar : ja
Alle übrigen Attribute: umgebungsabhängig

2.3 Identifizierung von Objekten

Die einzelnen Objekte können identifiziert werden über ihren vom Anwender bestimmbaren Namen. LLGL kennt folgende Sondernamen:

1 bezeichnet das zuletzt (durch einen START-Befehl, siehe Seite) geöffnete und noch nicht abgeschlossene höhere Objekt

O bezeichnet die Menge aller noch zu generierenden Objekte. Damit können für die Attribute Defaultwerte gesetzt werden.

Dem Anwender stehen damit als Namen alle positiven ganzen Zahlen größer 1 und kleiner einer oberen Grenze zur Verfügung. Diese obere Grenze liegt zur Zeit bei 66.

Außer über ihren Namen können Objekte auch identifiziert werden, indem man mit dem Lichtgriffel auf sie zeigt. Das Ergebnis dieser Zeigefunktion ist der Name, falls das Objekt bereits benannt worden ist, oder ein eindeutiger, von LLGL vergebener Pseudoname.

3. Der Befehlssatz von LLGL

Der Befehlssatz von LLGL zerfällt in drei Teilmengen:

- Befehle, die sichtbare Ausgabeobjekte erzeugen, löschen oder modifizieren, und
- Befehle zur Behandlung von Eingaben von einem der Eingabegeräte
- Sonderbefehle (Initialisierung, Hardcopy)

3.1 Befehle für die Bildausgabe

Bei den Befehlen, die die Ausgabeobjekte behandeln, unterscheidet man vier Gruppen: Befehle die Objekte

- generieren,
- löschen,
- erweitern oder
- Attribute von Objekten setzen

Generieren von Objekten

Zum Generieren von primitiven Objekten gibt es folgende Befehle

Poly2, Poly3	Erzeugt ein Polygon
Pol2li, Pol3li	Erzeugt eine Parameterliste für ein Polygon
Pol2na, Pol3na	Erzeugt ein Polygon unter Benutzung einer zuvor erzeugten Parameterliste
Progin	Erzeugt eine Programmierte Eingabe
Ref	Erzeugt eine Referenz auf ein höheres, referenzierbares Objekt
Vekto2, Vekto3	Erzeugt einen Vektor
Vek2li, Vek3li	Erzeugt eine Parameterliste für einen Vektor
Vek2na, Vek3na	Erzeugt einen Vektor unter Benutzung einer zuvor generierten Parameterliste

String	Erzeugt einen Symbolstring
Strili	Erzeugt eine Parameterliste für einen Symbolstring
Strina	Erzeugt einen Symbolstring unter Benutzung einer zuvor generierten Parameterliste

Höhere Objekte werden mit den folgenden Befehlen generiert:

Start	Eröffne ein höheres Objekt
End	Schließe ein höheres Objekt ab

Löschen von Objekten

Del	Löscht ein beliebiges Objekt

Erweitern von Objekten

Extend	Eröffnet ein höheres Objekt zur Erweiterung
Endext	Schließt die Erweiterung eines höheren Objekts ab

Attribute setzen

Setzen der Attribute, die der Selektion von Objekten dienen.

Lightp	Macht ein Objekt für den Lichtgriffel (un)sichtbar
Name	Benennt ein Objekt mit einem Benutzernamen

Setzen der Attribute, die das Aussehen von Vektoren oder Symbolen ändern

Blink	Läßt Objekte blinken
Underl	Läßt die Unterstreichungszeichen in einem Text blinken
Vectex	Setzt die Textur von Vektoren
Symtex	Setzt die Textur von Symbolen

Setzen der Attribute, welche die Topologie eines Objekts beeinflussen

Displa	Festlegen einer Translation
Vecsca	Setzt den Verkleinerungsfaktor für Vektoren
Symsca	Setzt den Verkleinerungsfaktor für Symbole
Symdir	Ändern der Schreibrichtung von Texten
Symref	Rotieren von Symbolen

Setzen der Attribute, die zur Verwaltung des Bildraumes verwendet werden

Units	Definieren der Einheitlängen auf den Achsen des Rasterkoordinatensystems
Viewp	Festlegen eines Bildausschnitts
Visual	Macht ein Bild (un)sichtbar

3.2 Befehle für die Eingabe

Die Befehle für die Behandlung von Eingaben des Sichtgerätes können unterteilt werden in

- Eingabedefinitionen und
- Eingabeverarbeitende Befehle

Eingabedefinitionen

ADEF	definiert die Zuordnung einer Eingabe zu einem Programm (Task)
AKILL	löscht die Zuordnung

Eingabeverarbeitende Befehle

ASTATE	ermöglicht die synchrone Abarbeitung von Eingaben durch Abfrage auf ein Eingabeereignis (Polling)

AWAIT gestattet die synchrone Abarbeitung von Eingaben durch Warten auf ein Eingabeereignis, sowie die asynchrone prioritätsgesteuerte Abarbeitung mit Hilfe von Attention-Tasks (nur in einer Multi-Task-Umgebung)

3.3 Sonderbefehle

Es extistieren drei Sonderbefehle:

GPON Schaltet das Bildschirmgerät ein und initialisiert das der Task zugeordnete Bild

GPOFF Löscht das der Task zugeordnete Bild und schaltet das Bildschirmgerät ab

HACO Erzeugt eine Hardcopy der Ausgabe auf dem Bildschirmgerät

4. Beschreibung der Befehle

ADEF (TYP, ADATA, COND, OPT)

Mit ADEF wird die Behandlung einer Eingabe durch ein Programm (Task) definiert. Die Eingabe eines Eingabegerätes wird dadurch einem Programm (Task) fest zugeordnet.

TYP	Bezeichnung des Eingabegerätes (s.u.)
ADATA	Array (bzw. Variable), in welchem die zu einer Eingabe gehörenden Daten abgelegt werden sollen (s.u.)
COND	Bedingung, der eine Eingabe unterworfen wird. Nur wenn die Bedingung erfüllt ist, wird die Eingabe an das Programm weitergegeben (s.u.)
OPT	Optionen für eine Eingabe (s.u.)

Wertevorrat der Parameter von ADEF:

Eingabegerät	Typ (Integer)	ADATA (Integer Array)	COND (Integer)	OPT (Integer)
Programmierte Eingabe	Ø	-Wert von PROGIN (Integ.) -X, -Y, -Z } physik.Koordinaten d.akt.Strahlposit. (Integer)	Ø:keine Bed. i(≠Ø): Eingabe nur,wenn Wert = i	Ø
Funktionstastatur	1	- Wert der gedrückten Taste (integer 1-32)	Ø:keine Bed. i≠Ø:Eingabe nur,wenn Wert der Taste = i	Ø
Lichtgriffel	2	- Menüzähler (Integer) - Anzahl k der Hierarchiestufen des gezeigten Objekts - R, - X } k-mal nach aufsteigenden Hierarchiestufen geordnet	Ø:keine Bed. i≠Ø:Eingabe nur,wenn Objektname auf der untersten Hierarchiestufe = i	Ø:Picking 1:Tracking
Alphanumerische Tastatur	4	- ASCII-Code der gedrückten Taste rechtsbündig,linkes Byte=Ø	Ø:keine Bed. i≠Ø:Eingabe nur, wenn ASCII-Code der Task=i	Ø

für R, X gilt:

<u>für Hierarchiestufe 0:</u> (niedrigste Hierarchiestufe):

a) k > 0

<u>R = 0</u>

X = Objektname (nicht referenzierbares, höheres Objekt),
Objektname (primitives Objekt)

<u>R = -1</u>

X = Objektname (referenzierbares, höheres Objekt),
Objektname (primitives Objekt)

b) k = 0 (nur 1 Hierarchiestufe vorhanden)

R = nicht vorhanden
X = Objektname (primitives Objekt)

<u>für Hierarchiestufen > 0:</u>

<u>R = Ø</u>

X = Objektname (nicht referenzierbares, höheres Objekt)
oder falls für nächstniedrigere Hierarchiestufe R=-1 ist
X = Objektname des primitiven Objekts REF, welches das auf der nächstniedrigeren Hierarchiestufe liegende referenzierbare, höhere Objekt aufgerufen hat

<u>R = -1</u>

X = Objektname (referenzierbares, höheres Objekt)

<u>Bemerkungen:</u>

1) Die Definition einer Eingabebehandlung mit ADEF muß <u>vor</u> dem 1. Eingabebefehl (AWAIT, ASTATE) erfolgt sein.

2) Für ein Eingabegerät ist jeweils die letzte erfolgte Eingabedefinition gültig.

3) Wird eine Eingabedefinition für ein Eingabegerät geändert, so werden evtl. noch nicht abgearbeitete Eingaben von diesem Gerät nur dann übernommen, wenn die Parameter COND und OPT der betreffenden Eingabedefinitionen übereinstimmen.

4) Das Eingabefeld ADATA muß so groß definiert werden, daß es Platz für alle aktuell auftretenden Eingabedaten bietet.

Beispiel:

```
DIMENSION LPFELD (10)
CALL ADEF (2, LPFELD, 15, 1)
```

Definition der Eingabebehandlung für den Lichtgriffel. Die Eingabedaten sollen im Array LPFELD stehen. Nur das Objekt mit dem Namen 15 bewirkt eine Eingabe. Der Lichtgriffel arbeitet im Tracking-Modus.

AKILL (TYP)

Hebt die Zuordnung einer Eingabe zu einem Programm auf.

TYP Bezeichnung des Eingabegerätes

Bemerkung:

1. Noch nicht abgearbeitete Eingaben werden verworfen
2. Die Task in der die zu löschende Definition mit ADEF gesetzt wurde, muß noch im System existent sein

Beispiel:

CALL AKILL (1)

Die Zuordnung der Funktionstastatureingabe zu diesem Programm wird aufgehoben.

ASTATE (TYP, SUB)

Eingabebefehl: Es wird auf eine Eingabe abgefragt (Polling). Liegt keine Eingabe vor, wird das Programm mit dem nächsten Befehl fortgesetzt; liegt eine Eingabe vor, wird zunächst die Subroutine SUB ausgeführt.

TYP Bezeichnung des Eingabegerätes
SUB Name einer Subroutine (welche die Eingabe verarbeitet)

Bemerkung:

1. Um die Eingabedaten in der Subroutine SUB verarbeiten zu können, muß das Eingabedatenfeld ADATA (siehe ADEF) in einem COMMON-Bereich liegen.

   Beispiel:

```
C    Hauptprogramm
     EXTERNAL FUTSUB, LPSUB, ANKSUB
     COMMON /C1/IVAR1(4)/C2/IVAR2(10) /C3/IVAR4
     CALL ADEF(1, IVAR1, Ø, Ø)
     CALL ADEF(2, IVAR2, Ø, Ø)
     CALL ADEF(4, IVAR4, Ø, Ø)
C    ABFRAGE AUF EINGABE (POLLING)
 10  CALL ASTATE (1, FUTSUB)
     CALL ASTATE (2, LPSUB)
     CALL ASTATE (4, ANKSUB)
          :
          :
     GOTO 10
          :
C    EINGABEVERARBEITENDE UNTERPROGRAMME
     SUBROUTINE FUTSUB
     COMMON /C1/FUTDAT(4)
          :
     Eingabeverarbeitung Funktionstastatur
          :
     RETURN
```

```
SUBROUTINE LPSUB
COMMON /C2/LPDAT(10)
      :
      :
Eingabeverarbeitung Lichtgriffel
      :
      :
RETURN

SUBROUTINE ANKSUB
COMMON /C3/ANKDAT
      :
      :
Eingabeverarbeitung alphanumerische Tastatur
      :
      :
RETURN
```

AWAIT (TYP)

Eingabebefehl: Das Programm wartet auf die Eingabe und wird bei Eintreffen der Eingabe mit dem nächsten Befehl fortgesetzt.

TYP Bezeichnung des Eingabegerätes

Bemerkung:

1. AWAIT wird in einer FORTRAN-Umgebung (Single-Task) wie eine READ-Anweisung benutzt (Synchrone Eingabebehandlung).
2. In einer Multi-Task-Umgebung kann AWAIT zum Aufbau von Attention-Tasks dienen, die eine asynchrone Behandlung der Eingaben ermöglichen. Jeder Eingabe kann eine Task mit verschiedener Priorität zugeordnet werden. Die Priorität der Attention-Tasks muß größer als die Priorität der Hauptprogramm-Task sein.
3. Die Verwaltung der Attention-Tasks (Aktivierung, Deaktivierung, Synchronisation mit anderen Tasks) wird nicht von LLGL übernommen, sondern mit den entsprechenden Primitiven des Betriebssystems (hier VORTEX) durchgeführt.
4. Vor der Deaktivierung einer Attention-Task durch EXIT (durch die Attention-Task selbst) oder ABORT (durch die Hauptprogrammtask) muß die zugehörige Eingabedefinition durch AKILL gelöscht werden.

Beispiel:

a) Anwendung von AWAIT zur synchronen Eingabebehandlung

```
C  HAUPTPROGRAMM
         :
         :
   CALL ADEF (4, IVAR, 0, 0)
         :
         :
C  EINGABE
   CALL AWAIT (4)
         :
         :
   END
```

b) Anwendung von AWAIT zur asynchronen Eingabebehandlung (siehe VORTEX-Betriebssystem-Handbuch).
Start des Hauptprogramms von der Konsole:

```
;SCHED, HPTASK, 4, 106, F
Task 1 HPTASK

C  HAUPTPROGRAMM
C  AKTIVIERUNG DER ATTENTION-TASKS
   CALL SCHED (10, 0, 106, 2H F, 6HLPTASK)
   CALL SCHED ( 7, 0, 106, 2H F, 6HANTASK)
              :
              :
C  DEAKTIVIERUNG DER ATTENTIONTASKS VOM
C  HAUPTPROGRAMM AUS
60 CALL AKILL (2)
   CALL ABORT (6HLPTASK)
   CALL AKILL (4)
   CALL ABORT (6HANTASK)

   Task 2 LPTASK

C  ATTENTIONTASK LIGHTPEN
   DIMENSION IVAR (12)
   CALL ADEF (2, IVAR, 0, 0)
10 CALL AWAIT (2)
              :
              :
   Eingabeverarbeitung Light-Pen
              :
              :
   GOTO 10

C  DEAKTIVIERUNG DER ATTENTIONTASK AUFGRUND
C  EINER BEDINGUNG IN DER EINGABEVERARBEITUNG
80 CALL AKILL (2)
   CALL EXIT
   END
```

```
   Task 3 ANTASK
C  ATTENTIONTASK ALPHANUM. TASTATUR
   CALL ADEF (4, IVAR, 0, 0)
10 CALL AWAIT (4)
             :
             :
   Eingabeverarbeitung alphanum. Tastatur
             :
             :
   GOTO 10
C  DEAKTIVIERUNG DER ATTENTIONTASK AUFGRUND
C  EINER BEDINGUNG IN DER EINGABEVERARBEITUNG
50 CALL AKILL (4)
   CALL EXIT
   END
```

BLINK (NAME, WERT)

BLINK setzt das Attribut Blinken.

NAME Name des Objekts, in dem das Attribut gesetzt werden soll (integer)

WERT Wert des Attributs (integer)

<O Blinke nicht
=O Umgebungsabhängig
>O Blinke

Beispiel:

CALL BLINK (13, 1)
läßt das Objekt mit Namen 13 blinken.

DEL (NAME)

DEL löscht ein Objekt

NAME Name des Objekts,das gelöscht werden soll

Beispiel:

CALL DEL (1)

löscht das aktuell offene Objekt

Bemerkungen:

1. Höhere, referenzierbare Objekte und Parameterlisten können nur gelöscht werden, wenn sie an keiner Stelle mehr referenziert werden. Im anderen Fall wird der Aufruf von DEL mit einer Fehlermeldung quittiert.
2. Soll ein Objekt gelöscht werden, das das aktuell offene Objekt enthält oder Teilobjekt des aktuell offenen Objekts ist, wird der Aufruf von DEL mit einer Fehlermeldung quittiert.

DISPLA (NAME, ABS, DX, DY, DZ)

Setzt das Attribut Translation (Displacement). Dadurch werden alle Koordinaten eines Objekts einer Verschiebung um einen von (DX, DY, DZ) abhängigen Vektor unterworfen.

NAME — Name des Objekts dessen Attribut gesetzt werden soll (integer)

DX,DY,DZ — Translation in x-, y- und z-Richtung (real). Die Werte beziehen sich auf die durch UNITS festgelegten Einheitslängen

ABS — Art der Translation

- <0 absolut
- =0 umgebungsabhängig
- >0 relativ

ABS beeinflußt die Berechnung des Translationsvektor (d_{neu}) in folgender Weise:

abs <0, dann $d_{neu} = (dx, dy, dz)$

abs >0, dann $d_{neu} = scale_{alt}(dx,dy,dz)+d_{alt}$

wobei d_{alt} = Translation des übergeordneten Objekts

$scale_{alt}$ = Skalierungsfaktor des übergeordneten Objekts

Beispiel:

Es existiert ein Objekt 3, ein Objekt 4 und ein aus diesen beiden Objekten zusammengesetztes Objekt 2.

Nach REF (2)

```
Units (8, -8, 0)
DISPLA (2, 1, 3.0, 0.0, 1.0)
```

erscheint die Ausgabe 1.
(Die Translation in z-Richtung ist ohne Wirkung, da die Einheit auf der z-Achse 0 ist)

Nach DISPLA (4, 1, -3.5, -2.0, 0) erhält man dann die Ausgabe 2.
Objekt 4 wurde nach oben verschoben, da die Einheit auf der y-Achse negativ ist.

Nach DISPLA (4, -1, -3.5, -2.0, 0) wird dann Ausgabe 3 ausgegeben.

EXTEND (NAME)

Mit EXTEND wird ein bereits abgeschlossenes höheres Objekt neu eröffnet, so daß es erweitert werden kann.

NAME Name des zu erweiternden Objekts

Bemerkungen:

1. Das zu erweiternde Objekt darf weder das aktuell offene sein, noch darf es in dem aktuell offenen enthalten sein oder es selbst enthalten.
2. Die Namen 1 und O sind hier nicht zugelassen.
3. Es darf zu einer Zeit nur ein Objekt erweitert werden. Zwischen zwei EXTEND-Befehlen muß immer ein ENDEXT erfolgen.

ENDEXT

Schließt das zum Erweitern geöffnete Objekt wieder ab.

END

END schließt das innerste, offene Objekt ab. Dieses Objekt wurde durch einen START-Befehl geöffnet.

(siehe START-Befehl)

GPOFF

Mit GPOFF wird das Sichtgerät abgeschaltet.

Bemerkungen:

1. Mit dem Absetzen von GPOFF verschwinden alle auf dem Bildschirm dargestellten Elemente.
2. Wird GPOFF vergessen, läuft das Displaygerät weiter bis zu einem GPON in einer anderen Task oder bis zur nächsten Initialisierung des Rechners

GPON

Mit GPON wird das Sichtgerät eingeschaltet und systeminterne Tabellen initialisiert.

Bemerkungen:

1. GPON ist die einzige, aber auch zwingend vorgeschriebene Initialisierungsroutine.
2. Wird GPON vergessen, befindet sich LLGL in einem undefinierten Zustand.

HACO

Mit HACO wird eine Hardcopy initialisiert. Das auf dem Schirm dargestellte Bild wird über den STATOS-Drucker auf Papier kopiert.

Bemerkungen:

1. Während des Kopiervorgangs sollten keine Änderungen des Bildes vorgenommen werden.
2. Die Papierkopie unterscheidet sich vom Schirmbild in zwei Punkten:

   - Es werden nur zwei Symbolgrößen dargestellt
   - Alle Elemente haben die gleiche Helligkeit (hier Schwärzungsgrad)

LIGHTP (NAME, WERT)

LIGHTP setzt das Attribut Lichtgriffel und legt damit fest, ob ein Objekt vom Lichtgriffel gesehen werden kann oder nicht.

NAME Name des Objekts, in dem das Attribut gesetzt werden soll (integer)

WERT Wert des Attributs (integer)
- <O unsichtbar
- =O umgebungsabhängig (Defaultwert)
- >O sichtbar

Beispiel:

```
CALL LIGHTP (3, 1)
```

macht das Objekt mit Namen 3 für den Lichtgriffel sichtbar

NAME (ALT, NEU)

NAME setzt das Attribut Name und erlaubt damit eine Änderung des Namens eines Objekts

ALT Name des Objekts, dessen Name geändert werden soll (integer)

NEU Neuer Name, den das Objekt erhalten soll (integer)

ALT kann ein beliebiger (auch Pseudo-) Name sein. NEU dagegen ist aus dem Bereich der gültigen, noch nicht verwendeten Benutzernamen zu wählen (siehe 2.3).

Beispiel:

```
CALL NAME (0, 2)
```
legt den Namen 2 für das nächste zu generierende Objekt fest

```
CALL NAME (2, 3)
```
benennt das Objekt mit dem Namen 2 um.

Bemerkungen:

1. Der Pseudonamen 0 hat für NAME eine gegenüber den anderen, Attribute setzenden Befehlen eingeschränkte Gültigkeitsdauer. Während er bei anderen Attributbefehlen für alle nachfolgend generierten Objekte gilt, gilt er hier nur für das nächste Objekt.
2. Name 1 bezeichnet wie bei allen Objekten das aktuell offene Objekt.

POL2LI (X,Y,Z,HD,LAENGE)

Mit POL2LI wird eine Parameterliste für einen Polygonzug, der in der x-y-Ebene liegt, erzeugt. Der so definierte Polygonzug wird aber erst sichtbar, wenn er durch POL2NA referenziert wird.

Bedeutung der Parameter siehe bei POLY2

Bemerkungen:

1. Vor dem Aufruf von POL2LI muß ein Aufruf von NAME erfolgen, um der zu generierenden Parameterliste einen Namen zu geben (durch CALL NAME(O,<Name der Parameterliste>)).
2. Die Arrays können maximal 4O94 Elemente groß sein
3. Weitere Hinweise siehe Bemerkungen zu POLY2

POL2NA (NAME)

Mit POL2NA wird ein Polygon ausgegeben unter Benutzung einer zuvor mit POL2LI erstellten Parameterliste.

NAME Name der zu referenzierenden Parameterliste (integer)

Bemerkungen:

1. Der Benutzer muß dafür sorgen, daß die referenzierte Parameterliste mit POL2LI erstellt wurde, da sonst für eine korrekte Ausgabe nicht garantiert werden kann.
2. Eine Parameterliste kann von maximal 7 POL2NA-Befehlen referenziert werden.

POL3LI (X,Y,Z,HD,LAENGE)

Mit POL3LI wird eine Parameterliste für einen Polygonzug erstellt. Sichtbar wird der Polygonzug aber erst, wenn er durch POL3NA referenziert wird.

Bedeutung der Parameter siehe bei POLY3

Bemerkungen:

1. Vor dem Aufruf von POL3LI muß ein Aufruf von NAME erfolgen, um der zu generierenden Parameterliste einen Namen zu geben (durch CALL NAME(0,<Name der Parameterliste>)).
2. Die Arrays können maximal 2730 Elemente lang sein.
3. Weitere Hinweise siehe Bemerkungen zu POLY3.

POL3NA (NAME)

Mit POL3NA wird ein Polygonzug ausgegeben unter Benutzung einer mit POL3LI erstellten Parameterliste

NAME Name der zu referenzierenden Parameterliste (integer).

Bemerkungen:

1. Der Benutzer muß sicherstellen, daß die referenzierte Parameterliste mit POL3LI erstellt wurde, da sonst für eine korrekte Ausgabe nicht garantiert werden kann.
2. Eine Parameterliste kann bis zu siebenmal referenziert werden.

POLY2 (X,Y,Z,HD,LAENGE)

Mit POLY2 kann ein Polygonzug ausgegeben werden, der in der x-y-Ebene des Koordinatensystems liegt.

X,Y Array der x- und y-Koordinaten der Stützpunkte des Polygons (real)
Z Gemeinsame Z-Koordinate aller Stützpunkte (real)
HD Array das angibt, ob die Verbindung zu dem Stützpunkt j hin sichtbar (HD(j)≠0) oder unsichtbar (HD(j)=0) gezeichnet werden soll (integer)
LAENGE Gemeinsame Länge der vorkommenden Arrays (integer)

Beispiel:

```
REAL X(3)/-0.5, 1.0, 1.0/Y(3)/1.0, 1.0, 2.0/
INTEGER HD(3)/0, 1, 1/
CALL POLY2 (X,Y,0,HD,3)
```

generiert folgende Ausgabe:

Bemerkungen:

1. Der nullte Stützpunkt des Polygons ist die aktuelle Position des Elektronenstrahls, d.h., wenn hd(1)=1, dann wird von der aktuellen Position zu (X(1),Y(1),Z) eine sichtbare Verbindung gezogen.
   Die aktuelle Position ist gegeben als
   - undefiniert, wenn von der POLY2 aufrufenden Task bisher keine Ausgabe erzeugt wurde
   - sonst der Endpunkt des zuletzt von der Task erzeugten Elements; bei Vektoren ist das der Endpunkt, bei Polygonen der letzte Stützpunkt, bei Texten der letzte Buchstabe im Text.
2. Die Koordinaten werden als Gleitpunktzahlen angegeben und beziehen sich auf die durch UNITS definierten Einheitenlängen.

POLY3 (X,Y,Z,HD,LAENGE)

Mit POLY3 kann ein Polygonzug ausgegeben werden.

X,Y,Z Arrays mit x-, y- und z-Koordinaten der Stützpunkte (real)

HD Array, das angibt, ob die Verbindung zum Stützpunkt j sichtbar (HD(j)≠0) oder unsichtbar (HD(j)=0) gezeichnet werden soll (integer)

LAENGE Gemeinsame Länge der auftretenden Arrays (integer)

Bemerkungen:

1. siehe Bemerkungen zu POLY2
2. Benutzung nur mit GP 430/440 sinnvoll.

PROGIN (WERT)

Mit PROGIN wird ein 'Programmierte Eingabe'-Objekt erzeugt.

WERT Ein Wert der bei Eintritt des Eingabeereignisses übergeben wird (integer)

Beispiel:

```
CALL PROGIN(2)
```

REF (NAME)

Mit REF wird ein höheres, referenzierbares Objekt sichtbar gemacht; es wird ein Unterbild aufgerufen.

NAME Name des zu referenzierenden Objekts

Bemerkungen:

1. Die Anzahl der Referenzen ist nicht beschränkt.
2. Durch die Referenz wird ein Objekt nur sichtbar gemacht. Seine Größe und Lage wird nicht beeinflußt. Verschieben und verkleinern läßt sich das Objekt über die Attribute Translation und Vektor-Symbolskalierung des aufrufenden Referenzobjekts (Befehle DISPLA, VECSCA und SYMSCA)

Beispiel:

```
Mit     CALL NAME (0,2)
        CALL START (1)
        CALL VEKTO2 (0.0, 0.0, 1.0, 10.0, 10.0)
        CALL END
```

wird ein referenzierbares, höheres Objekt mit dem Namen 2 generiert. Es hat folgendes Aussehen.

```
Mit     CALL NAME (0,3)
        CALL REF (2)
        CALL DISPLA (3, 1, 0, -10.0, 0)
        CALL DISPLA (0, 1, 10.0, 10.0, 0.0)
        CALL VECSCA(0, 1, 0.5)
        CALL REF (2)
```

wird folgende Ausgabe erzeugt

START (SUB)

START eröffnet ein neues höheres Objekt. Alle zwischen START- und zugehörigem END-Befehl generierten Objekte sind Teilobjekte des durch START eröffneten.

SUB = 0 Generiere ein nicht referenzierbares, höheres Objekt
≠ 0 Generiere ein referenzierbares, höheres Objekt

Bemerkung:

1. Ein höheres, referenzierbares Objekt ist im Prinzip nur sichtbar, wenn es referenziert wird (siehe REF-Befehl). Während der Erstellungsphase, also zwischen START- und zugehörigem END-Befehl, wird das Objekt jedoch als höher, nicht referenzierbar behandelt, sodaß es während der Erstellungsphase ohne Referenz sichtbar ist.
2. START-/END-Befehlspaare können geschachtelt werden (ähnlich BEGIN-END oder SUBROUTINE-END in höheren Programmiersprachen) bis zu einer maximalen Tiefe von 10.
2. Das zuletzt durch einen START-Befehl geöffnete und noch nicht abgeschlossene Objekt kann mit dem Namen 1 referenziert werden. Dieses Objekt heißt aktuell offenes Objekt.

Beispiel:

```
CALL NAME (0,2)
CALL START (1)                    /* Beginn des Objekts 2*/
CALL VEKTO2 (X,Y,Z,Y,B)
CALL START (0)                    /* Beginn des Objekts 5*/
CALL NAME (1,5)
CALL VEKTO2 (100.0,3.5,0.0,0.0,0.0)
CALL END                          /* Ende des Objekts 5*/
CALL START (0)                    /* Beginn des Objekts 3*/
CALL NAME (1,3)
CALL NAME (0,4)
CALL START (0)                    /* Beginn des Objekts 4*/
CALL POLY2 (X,Y,Z,HF,L)
CALL END                          /* Ende des Objekts 4*/
```

```
CALL END                          /*Ende des Objekts 3*/
CALL STRING (C,L,O,X,Y,Z)
CALL END                          /*Ende des Objekts 2*/
```

baut ein höheres, referenzierbares Objekt mit dem Namen 2 auf, das zusammengesetzt ist aus einem Vektor, dem höheren, nicht referenzierbaren Objekt 5, dem höheren, nicht referenzierbaren Objekt 3 und einem Symbolstring. Objekt 3 besteht aus dem höheren nicht referenzierbaren Objekt 4, das seinerseits aus einem Polygon besteht. Objekt 5 besteht aus einem Vektor.

STRILI (CHARR, ANZ, POS, X, Y, Z)

Mit STRILI wird eine Parameterliste für einen Symbolstring erstellt. Sichtbar wird der String aber erst, wenn er durch STRINA referenziert wird.

Bedeutung der Parameter siehe bei STRING.

Bemerkungen:

1. Vor dem Aufruf von STRILI muß ein Aufruf von NAME erfolgen, um der zu generierenden Parameterliste einen Namen zu geben (durch CALL NAME (0,<Name der Parameterliste>)).
2. ANZ kann höchstens den Wert 8190, wenn POS=0, annehmen, 8187 sonst.
3. Weitere Hinweise siehe Bemerkungen zu STRING.

STRINA (NAME)

Mit STRINA wird ein Symbolstring ausgegeben unter Benutzung einer mit STRILI erstellten Parameterliste.

NAME Name der zu referenzierenden Parameterliste (integer)

Bemerkungen:

1. Der Benutzer muß sicherstellen, daß die referenzierte Parameterliste mit STRILI erstellt wurde, da sonst für eine korrekte Ausgabe, auch der vorher erstellten Ausgabe, nicht garantiert werden kann.
2. Eine Parameterliste kann bis zu siebenmal referenziert werden.

STRING (CHARR, ANZ, POS, X, Y, Z)

Mit String kann ein Symbolstring ausgegeben werden.

CHARR Array, in dem die Characters des auszugebenden Strings abgelegt sind (integer)

ANZ Länge des Arrays (integer)

POS ≠0 positioniere den Text an den Punkt (x,y,z)
=0 positioniere den Text an die aktuelle Position (siehe Bemerkungen zu POLY2)
(integer)

X,Y,Z Wenn POS≠0, die x-, y- und z-Koordinate des Punktes, an dem der Text stehen soll. Die Koordinaten beziehen sich auf die durch UNITS festgelegten Einheitenlängen (real)
Wenn POS=0, dann ohne Bedeutung

Bemerkungen:

1. In CHARR wird der auszugebende String mit einem oder zwei Zeichen pro Arrayelement abgelegt. Der Zeichenvorrat ist in der Tabelle 2 dargestellt. Die dort aufgeführten Zeichen ab Code 160 bis Code 223 sind in FORTRAN als Textkonstanten definiert.

2. Das Zeichen VT (Code 139) darf vom Benutzer nicht verwendet werden (der Displayprozessor benötigt dieses Zeichen als Endekennung eines Strings).

3. Nach jedem Zeichen außer dem letzten wird automatisch ein Zeichenvorschub eingefügt.

4. Die Zeichen LF und CR (Code 138 und 141) bewirken einen Zeilenvorschub bzw. Wagenrücklauf. Über das Attribut Schreibrichtung wird die Richtung bestimmt in der Vorschub und Rücklauf gemacht werden. In jedem Fall beziehen sie sich auf die Position des ersten Zeichens in dem String

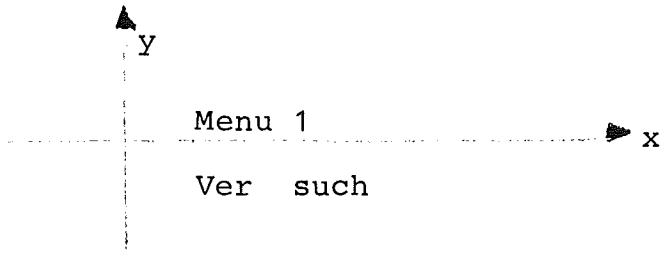
5. Mit dem Zeichen FF (Code 140) können Menüs aufgebaut werden. Jedesmal wenn ein solches Zeichen ausgegeben wird, wird der Menüzähler inkrementiert. Dieser Zähler wird bei einer Eingabe vom Lichtgriffel übergeben (siehe ADEF).
   Jedem Objekt Bild ist ein solcher Menüzähler zugeordnet. Bei leerem Bild hat er den Wert 1 und wird mit jedem Zeichen FF um 1 erhöht. Die Inkrementierungen kumulieren sich, der Zähler wird also nicht vor jeder Ausgabe eines Symbolstrings initialisiert.

6. Gibt es keine Symbolskalierung für den auszugebenden Symbolstring, dann werden Symbolstrings in ihrer maximalen Größe von ca. 33 mm ausgezeichnet.

Beispiele:

1. CALL STRING ('TEXT1',3,0,0.0,0.0,0.0)
   schreibt an die aktuelle Position den Text TEXT1

2. DATA X/140,'M',229,238,245,' ',177,141,138,140,'V',229,242,
   ' ',140,243,245,227,232/
   CALL STRING (X,19,1,0.0,0.0,0.0)
   erzeugt folgende Ausgabe



   Sei dieser String das einzige Objekt in dem der Task zugehörigen Bild, dann liefert eine Eingabe von dem Lichtgriffel den Menüzähler 2, wenn auf einen der Buchstaben M,e,n,u oder 1
   3, wenn auf einen der Buchstaben V,e oder r und
   4, wenn auf einen der Buchstaben s,u,c oder h

   gezeigt wurde.

Code	ASCII	Display	Code	ASCII	Display	Code	ASCII	Display
128	NUL	NUL	172	,	,	216	X	X
129	SOH	✳	173	-	-	217	Y	Y
130	STX	┼	174	.	.	218	Z	Z
131	ETX	○	175	/	/	219	[	[
132	EOT	□	176	Ø	Ø	220	\	\
133	ENQ	→	177	1	1	221	]	]
134	ACK	↘	178	2	2	222	^	^
135	BEL	N.S.Underline	179	3	3	223	_	_
136	BS	☆	180	4	4	224		
137	HT	‖	181	5	5	225	a	a
138	LF	New Line	182	6	6	226	b	b
139	VT	End of String	183	7	7	227	c	c
140	FF	Increment Menu	184	8	8	228	d	d
141	CR	Carriage Return	185	9	9	229	e	e
142	SO	+	186	:	:	230	f	f
143	SI	⊕	187	;	;	231	g	g
144	DLE	△	188	<	<	232	h	h
145	DC1	Φ	189	=	=	233	i	i
146	DC2	φ	190	>	>	234	j	j
147	DC3	θ	191	?	?	235	k	k
148	DC4	π	192	∂	∂	236	l	l
149	NAK	μ	193	A	A	237	m	m
150	SYN	>	194	B	B	238	n	n
151	ETB	<	195	C	C	239	o	o
152	CAN	≠	196	D	D	240	p	p
153	EM	O (degree)	197	E	E	241	q	q
154	SS	¬	198	F	F	242	r	r
155	ESC	∉	199	G	G	243	s	s
156	FS	Σ	200	H	H	244	t	t
157	GS	Ω	201	I	I	245	u	u
158	RS	δ	202	J	J	246	v	v
159	US	ε	203	K	K	247	w	w
160	Space	Space	204	L	L	248	x	x
161	!	!	205	M	M	249	y	y
162	"	"	206	N	N	250	z	z
163	#	#	207	O	O	251	{	{
164	$	$	208	P	P	252	|	|
165	%	%	209	Q	Q	253	}	}
166	&	&	210	R	R	254	~	~
167	'	'	211	S	S	255	DEL	adage
168	(	(	212	T	T			
169	)	)	213	U	U			
170	*	*	214	V	V			
171	+	+	215	W	W			

Tabelle 2 Zeichencodes

Die linke Spalte enthält den Code als Dezimalzahl, die mittlere gibt die Bedeutung im ASCII-Code und die rechte Spalte zeigt die Darstellungsform auf dem Bildschirmgerät.

SYMDIR (NAME, WERT)

Mit SYMDIR wird das Attribut Schreibrichtung gesetzt und damit die Schreibrichtung für die Textausgabe geändert.

NAME Name des Objekts, in dem das Attribut geändert werden soll (integer)

WERT Wert des Attributs (integer)

0	umgebungsabhängig									
1	Zeilen	von	links	nach	rechts,	Linien	von	oben	nach	unten
2	"	"	links	"	rechts,	"	"	unten	"	oben
3	"	"	rechts	"	links,	"	"	oben	"	unten
4	"	"	rechts	"	links,	"	"	unten	"	oben
5	"	"	oben	"	unten,	"	"	links	"	rechts
6	"	"	oben	"	unten,	"	"	rechts	"	links
7	"	"	unten	"	oben,	"	"	links	"	rechts
8	"	"	unten	"	oben,	"	"	rechts	"	links

Beispiel:

Objekt 2 ergibt mit CALL SYMDIR (2,1) folgende Ausgabe:

Zeile 1
Zeile 2

Mit CALL SYMDIR (2,2) erhält man

Zeile 2
Zeile 1

CALL SYMDIR (2,4) ergibt

2elieZ
1elieZ

CALL SYMDIR (2,6) ergibt

```
Z       Z
e       e
i       i
l       l
e       e
2       1
```

CALL SYMDIR (2,7) ergibt

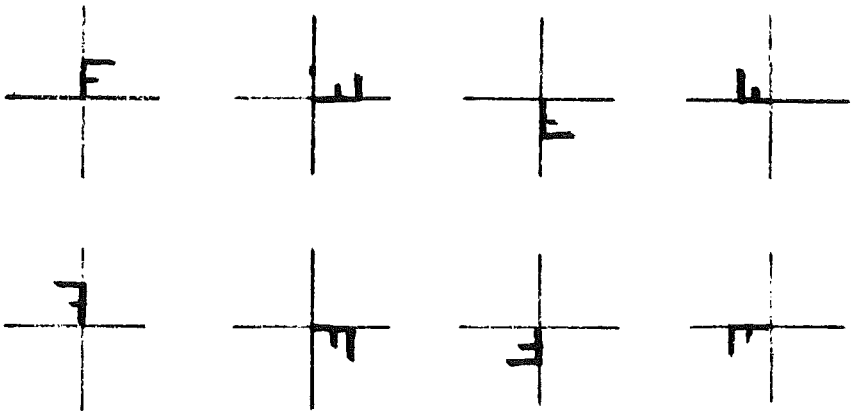
```
1       2
e       e
l       l
i       i
e       e
Z       Z
```

SYMREF (NAME, WERT)

Mit SYMREF wird das Attribut Symboldrehung gesetzt und damit ein Symbol um einen bestimmten Winkel gedreht.

NAME Name des Objekts, dessen Attribut gesetzt werden soll (integer)

WERT Wert des Attributs (integer)
O umgebungsabhängig
sonst (dargestellt am Buchstaben F)



Beispiel:

CALL SYMREF (O, 1)
setzt den Defaultwert der Symboldrehung auf den Wert 1

SYMSCA (NAME, ABS, WERT)

SYMSCA setzt das Attribut Symbolskalierung. Dadurch ist es möglich Symbole zu verkleinern.

NAME Name des Objekts, dessen Attribut gesetzt werden soll (integer)

ABS Art der Verkleinerung (integer)

=0 umgebungsabhängig
>0 relativ

ABS beeinflußt die Berechnung des neuen Skalierungsfaktors (s_{neu})

ABS = 0 $s_{neu} = s_{alt}$
> 0 $s_{neu} = s_{alt}$ WERT
mit s_{alt}=Skalierungsfaktor des eine Hierarchiestufe höher liegenden Objekts

WERT Skalierungsfaktor (real)

Beispiel:

```
CALL START (0)
CALL NAME (1,2)
CALL STRING ('AB',1,1,0.0,0.0,0.0)
CALL NAME (0,3)
CALL SYMSCA (0,1,0.5)
CALL STRING ('CD',1,0,0.0,0.0,0.0)
CALL END
```

baut ein nicht referenzierbares, höheres Objekt auf, das den Namen 2 hat. Dieses Objekt besteht aus zwei primitiven Objekten Symbolstring, wovon das zweite den Namen 3 hat. Auf dem Bildschirm wird folgendes ausgegeben:

ABCD

CALL SYMSCA (2,0,0.5) ändert die Ausgabe in

ABCD

CALL SYMSCA (3,1,1.0) macht daraus

ABCD

Bemerkungen:

1. WERT ist eine Realzahl zwischen 0.0 und 0.999. Hardwarebedingt sind nur 256 verschiedene Symbolgrößen erlaubt, so daß der Skalierungsfaktor systemintern quantisiert wird.
2. Die Skalierung betrifft nicht nur die Symbolgröße, sondern in gleicher Weise auch die Größe des Zeichenabstands und des Zeilenvorschubs.

SYMTEX (NAME, WERT)

SYMTEX setzt das Attribut Symboltextur und bestimmt damit die Textur,in der Symbole dargestellt werden.

NAME Name des Objekts,in dem das Attribut geändert werden soll (integer)

WERT Wert des Attributs (integer)

0 umgebungsabhängig

1 durchgezogene Linien

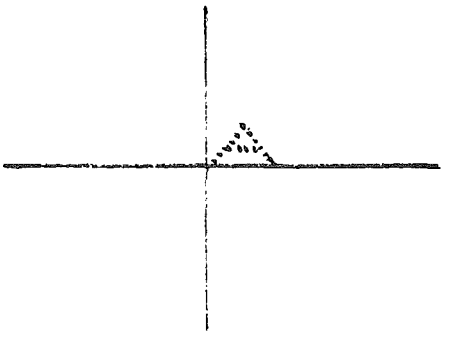
2 gepunktete Linien

3 gestrichelte Linien

Beispiel:

```
CALL SYMTEX (0,2)
CALL STRING ('A',1,0,0.0,0.0,0.0)
```

ergibt folgendes Bild



UNDERL (NAME, WERT)

UNDERL setzt das Attribut Unterstreichung, womit der Blinkmodus des Zeichens 'non spacing underline' (Code 135) geändert werden kann.

NAME Name des Objekts, in dem das Attribut gesetzt werden soll

WERT Wert des Attributs (integer)
<O Blinke nicht
>=O Blinke

Bemerkungen:

1. Das 'non spacing underline' blinkt unabhängig davon, ob das Objekt Symbolstring, in dem das Zeichen auftritt, blinkt oder nicht. Jedoch kann man das Blinken dieses Zeichens nicht mit CALL UNDERL (..., -1) verhindern, wenn es Teil eines Symbolstrings ist, der selber blinkt.
2. Das Attribut Unterstreichung ist nur primitiven Objekten zugeordnet. Es hat daher keinen Wert umgebungsabhängig in seinem Wertevorrat.

UNITS (XE, YE, ZE)

UNITS setzt das Attribut Einheiten. Damit wird die Länge der Einheitsvektoren auf dem Rasterkoordinatensystem festgelegt.

XE,YE,ZE Länge der Einheitsvektoren auf der x-, y- und z-Achse des Rasterkoordinatensystems. Wird angegeben durch die Anzahl der Adressen pro Einheitsvektor. (integer)

Beispiel:

CALL UNITS (800,800,0)
setzt die Einheitenlänge auf der x- und y-Achse auf 800 adressierbare Punkte und da 8 adressierbare Punkte immer auf den gleichen Rasterpunkt abgebildet werden (s. 1.2.1) zu 100 Rasterpunkten (≙ ca. 1.5 cm). Die Einheitslänge auf der z-Achse ist 0, d.h. alle zu generierenden Elemente liegen in der x-y-Ebene, die auch den Koordinatensprung enthält.

Bemerkungen:

1. Die Festlegung der Einheiten gilt nur für das Bild das der Task zugeordnet ist, aus der UNITS aufgerufen wurde. Auf Bilder, die anderen Task zugeordnet sind, hat dieser Aufruf keinen Einfluß.
2. Die Längenfestlegung gilt nur für Koordinaten, die zeitlich nach dem Aufruf von UNITS benutzt werden. Bereits definierte Objekte ändern ihre Gestalt nicht, wenn sie in einem Koordinatensystem mit einer anderen Länge, als der mit der sie definiert wurden, ausgegeben werden (Dafür sind die Befehle DISPLA, VECSCA, SYMSCA vorgesehen).
3. Eine negative Einheitslänge bedeutet eine Änderung der Achsenrichtung.

VECSCA (NAME, ABS, SCALE)

VECSCA setzt das Attribut Vektorskalierung. Dieses Attribut setzt einen Verkleinerungsfaktor, mit dem alle Koordinaten multipliziert werden. VECSCA verkleinert also Objekte (siehe Kapitel 1.2.1).

NAME Name des Objekts, dessen Attribut geändert werden soll (integer)

SCALE Skalierungsfaktor zwischen -1.0 und +1.0 (real)

ABS Art der Skalierung
<0 absolut
=0 umgebungsabhängig
>0 relativ

ABS beeinflußt die Bestimmung des neuen Skalierungsfaktors (s_{neu}) unter Verwendung von SCALE und dem Skalierungsfaktor des übergeordneten Objekts (s_{alt})

ABS<0, dann s_{neu}=SCALE
ABS=0, dann $s_{neu}=s_{alt}$
ABS>0, dann s_{neu}=SCALE*s_{alt}

Bemerkung:

1. Die Matrix S (siehe Transformationen in Kapitel 1.2.1) degeneriert

$$\text{zu } s_{neu} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ , d.h.}$$

alle Objekte werden einer zentrischen Streckung unterworfen. VECSCA bestimmt den Streckungsfaktor. Das Streckungszentrum ist der Ursprung des Rasterkoordinatensystems

2. Ein negativer Streckungsfaktor bedeutet eine zusätzliche Punktspiegelung um das Streckungszentrum.

3. Diese Streckung wirkt auf alle durch Koordinaten beschriebenen Objekte. Das heißt, die Charactergröße wird durch VECSCA nicht beeinflußt, wohl aber der Anfangspunkt eines Symbolstrings.

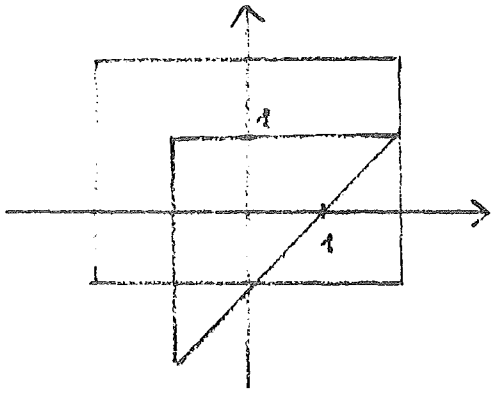
Beispiel:

Mit

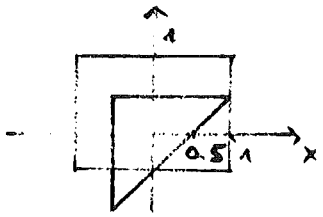
```
CALL UNITS (1,1,0)
DATA X/-2.0,2.0,2.0,-2.0,-2.0/,Y/-1.0,-1.0,2.0,2.0,-1.0/
     Z/0.0/,HD/0,1,1,1,1/
DATA A/-1.0,2.0,-1.0,-1.0/,B/-2.0,1.0,1.0,-2.0/,D/0.0/,
     HDA/0,1,1,1/
CALL START (1)
CALL NAME (1,2)
CALL POLY2 (X,Y,Z,HD,5)
CALL START (0)
CALL NAME (1,3)
CALL POLY2 (A,B,D,HDA,4)
CALL END
CALL END
CALL REF (2)
```

wird ein Objekt 2 referenziert, das aus einem Polygon und einem höheren, nicht referenzierbaren Objekt 3 zusammengesetzt ist. Objekt 3 besteht aus einem Polygon.

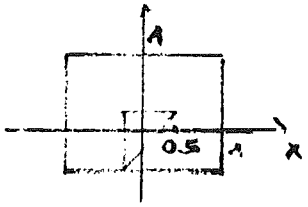
Auf dem Schirm wird ausgegeben



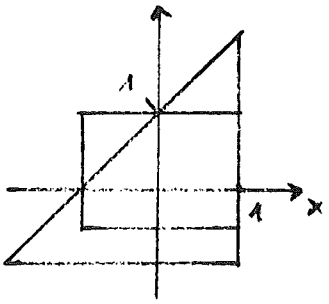
CALL VECSCA (2,1,0.5) macht daraus



CALL VECSCA (3,1,0.5) ergibt



CALL VECSCA (3, -1, -1.0) ändert das Bild in



VECTEX (NAME, WERT)

VECTEX setzt das Attribut Vektortextur und legt damit die Textur von Polygonen und Vektoren fest

NAME Name des Objekts, in dem das Attribut gesetzt werden soll (integer)

WERT Wert des Attributs (integer):

0 umgebungsabhängig
1 durchgezogene Linien
2 gepunktete Linien
3 gestrichelte Linien
4 Linien, von denen nur die Endpunkte sichtbar sind

Beispiel:

CALL VECTEX (0,1)
setzt den Defaultwert der Textur auf durchgezogene Linien

VEKTO2 (X1, Y1, Z, X2, Y2)

Mit VEKTO2 wird ein Vektor ausgegeben, der in der x-y-Ebene des Koordinatensystems liegt.

X1, Y1 Anfangs-
X2, Y2 End- koordinaten des Vektors (real)
Z Gemeinsame dritte Koordinate des Anfangs- und Endpunkts (real)

Beispiel:

```
CALL UNITS (8,8,0)
CALL VEKTO2 (-2.0, -2.0, 0.0, 2.0, 1.0)
```

generiert die Ausgabe

Bemerkung:

1. Die Koordinaten werden als Gleitpunktzahlen angegeben, die sich auf die durch UNITS definierten Einheitslängen beziehen.

VEKTO3 (X1, Y1, Z1, X2, Y2, Z2)

Mit VEKTO3 wird ein Vektor ausgegeben.

X1, Y1, Z1	Koordinaten des Anfangspunktes des Vektors (real)
X2, Y2, Z2	Koordinaten des Endpunktes des Vektors (real)

Bemerkungen:

1. Die Koordinaten beziehen sich auf die durch UNITS definierten Einheitslängen

VEK2LI (X1, Y1, Z, X2, Y2)

Mit VEK2LI wird eine Parameterliste für einen Vektor erzeugt, der in der x-y-Ebene des Koordinatensystems liegt. Der so definierte Vektor wird erst sichtbar, wenn er durch VEK2NA referenziert wird.

Bedeutung der Parameter siehe VEKTO2

Bemerkungen:

1. Vor dem Aufruf von VEK2LI muß ein Aufruf von NAME erfolgen, um der zu generierenden Parameterliste einen Namen zu geben (durch CALL NAME (O,<Name der Parameterliste>)).
2. Eine Parameterliste für einen Vektor ist nicht eine Liste von Vektoren, sondern eine Parameterliste mit Daten für einen einzigen Vektor.

VEK2NA (NAME)

Mit VEK2NA wird ein Vektor ausgegeben unter Benutzung einer zuvor mit VEK2LI erstellten Parameterliste.

NAME Name der zu referenzierenden Parameterliste (integer)

Bemerkungen:

1. Der Benutzer muß dafür sorgen, daß die referenzierte Parameterliste mit VEK2LI erstellt wurde, da sonst für eine korrekte Ausgabe nicht garantiert werden kann.
2. Eine Parameterliste kann von maximal 7 VEK2NA-Befehlen referenziert werden.

VEK3LI (X1, Y1, Z1, X2, Y2, Z2)

Mit VEK3LI wird eine Parameterliste für einen Vektor erstellt. Sichtbar wird der Polygonzug aber erst, wenn er durch VEK3NA referenziert wird.

Bedeutung der Parameter siehe VEKTO3.

Bemerkungen:

1. Vor dem Aufruf von VEK3LI muß ein Aufruf von NAME erfolgen, um der zu generierenden Parameterliste einen Namen zu geben (durch CALL NAME (0,<Name der Parameterliste>)).

VEK3NA (NAME)

Mit VEK3NA wird ein Vektor ausgegeben, unter Benutzung einer mit VEK3LI erstellten Parameterliste

NAME Name der zu referenzierenden Parameterliste (integer)

Bemerkungen:

1. Der Benutzer muß sicherstellen, daß die referenzierte Parameterliste mit VEK3LI erstellt wurde, da sonst für eine korrekte Ausgabe nicht garantiert werden kann.
2. Eine Parameterliste kann von maximal 7 VEK3NA-Befehlen referenziert werden.

VIEWP (XU, YU, LAENGE)

VIEWP setzt das Attribut Ausschnitt und legt damit den Ausschnitt des virtuellen Gerätebildraums fest, der auf den physikalischen Gerätebildraum abgebildet wird.
(siehe Kap. 1.2.1)

XU, YU Koordinaten der linken unteren Ecke des Ausschnittquadrats (integer)
LAENGE Halbe Seitenlänge des Ausschnittquadrats (integer)

Bemerkungen:

1. XU, YU, LAENGE werden angegeben in Bezug auf das Rasterkoordinatensystems (siehe Kapitel 1.2.1)
2. LAENGE ist größer gleich 8192. Wird ein kleinerer Wert angegeben, dann wird statt dessen 8192 eingesetzt. Der Ausschnitt kann also nicht kleiner werden als der physikalische Gerätebildraum.

VISUAL (WERT)

VISUAL setzt das Attribut Sichtbar und bestimmt damit, ob das der aufrufenden Task zugeordnete Bild sichtbar oder unsichtbar dargestellt wird.

WERT Wert des Attributs

<0 unsichtbar

>=0 sichtbar

Beispiel:

```
CALL VISUAL (-1)
```

macht alle Objekte eines Bildes unsichtbar.

5. Programmbeispiel

Anhand der Programmierung eines Anwendungsbeispiels sollen die Fähigkeiten von LLGL und die Möglichkeiten zum Aufbau eines interaktiven graphischen Programms gezeigt werden. Der besseren Übersicht wegen sind einige Routinen nicht voll ausgeführt.

Ein Benutzer des Anwendungsprogramms kann an einem Bildschirm mit Hilfe eines Lichtgriffels und dreier Funktionstasten einen Schaltkreis (z.B. die astabile Kippschaltung von Bild 5-1) aufbauen.

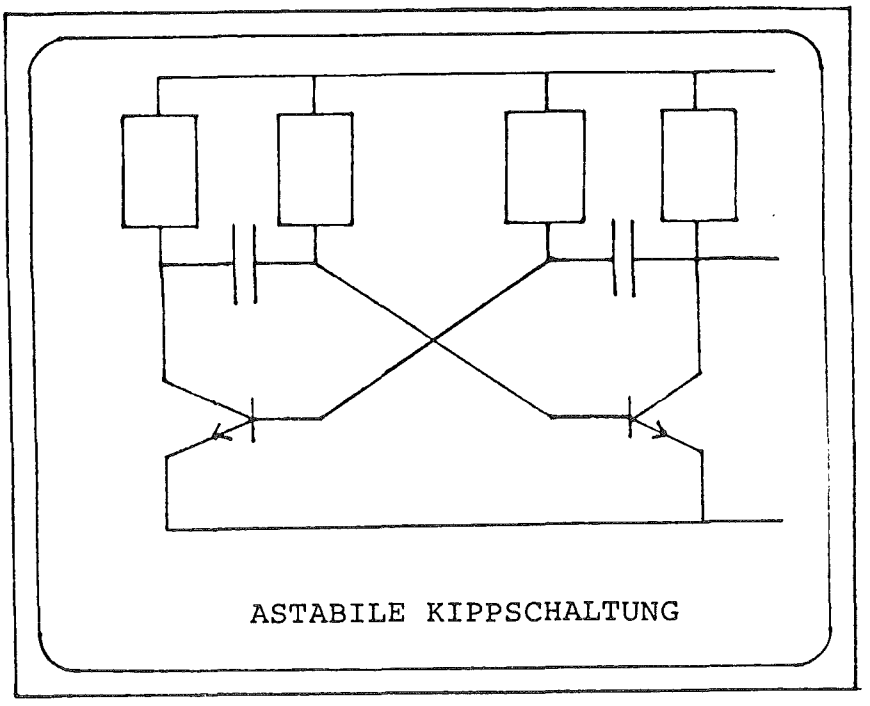


Bild 5-1: Bildausgabe: astabile Kippschaltung

Hierzu kann er aus einer Menüliste am linken Bildschirmrand ein Schaltkreiselement (Linie, Widerstand, Kondensator oder Transistor) mit dem Lichtgriffel auswählen. Das Programm fordert ihn durch eine Textausgabe am unteren Bildschirmrand zu entsprechenden Aktionen auf.

Bild 5-2 zeigt eine Momentaufnahme einer Entwurfsphase.

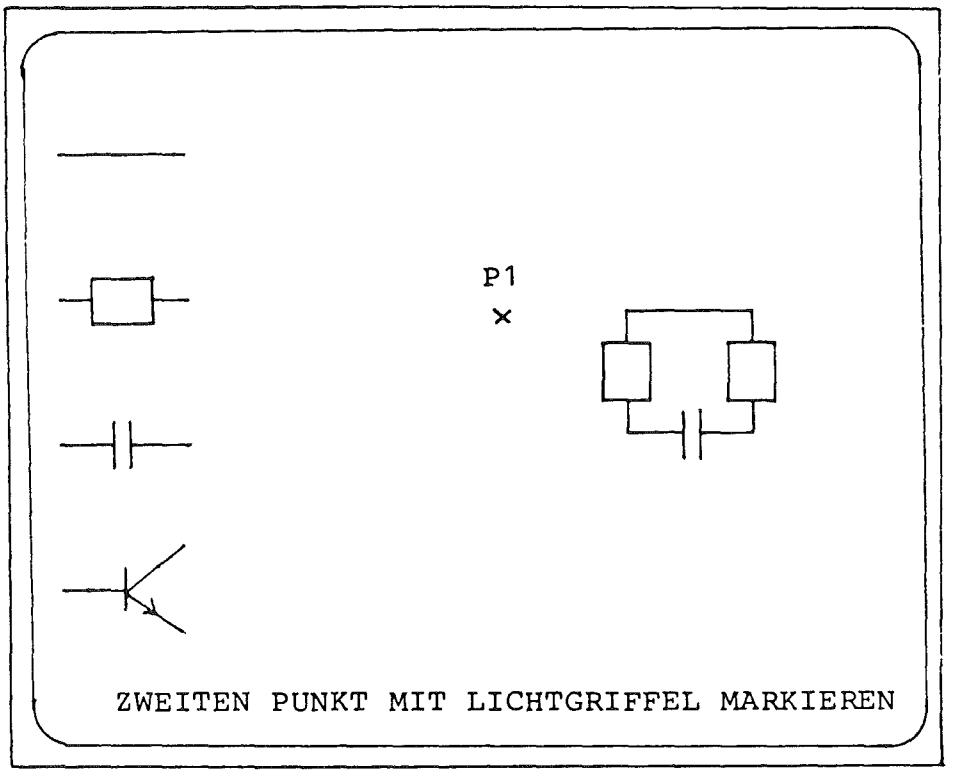


Bild 5-2: Momentaufnahme eines interaktiven Schaltkreisentwurfs

An Hand eines Ablaufdiagramms des Anwendungsprogramms wird der Entwurf einer Schaltung erläutert (Bild 5-3):

Das Programm gliedert sich in vier Teile:

1. Vereinbarungen, Initialisierungen,
   Definition der Schaltkreiselemente und der Menüliste,
2. Bild (Menüliste und Schaltbild)-Ausgabe,
   Synchronisation auf externe Benutzereingriffe,
3. Abarbeitung der Lichtgriffeleingabe, sowie
4. Abarbeitung der Funktionstastatureingabe.

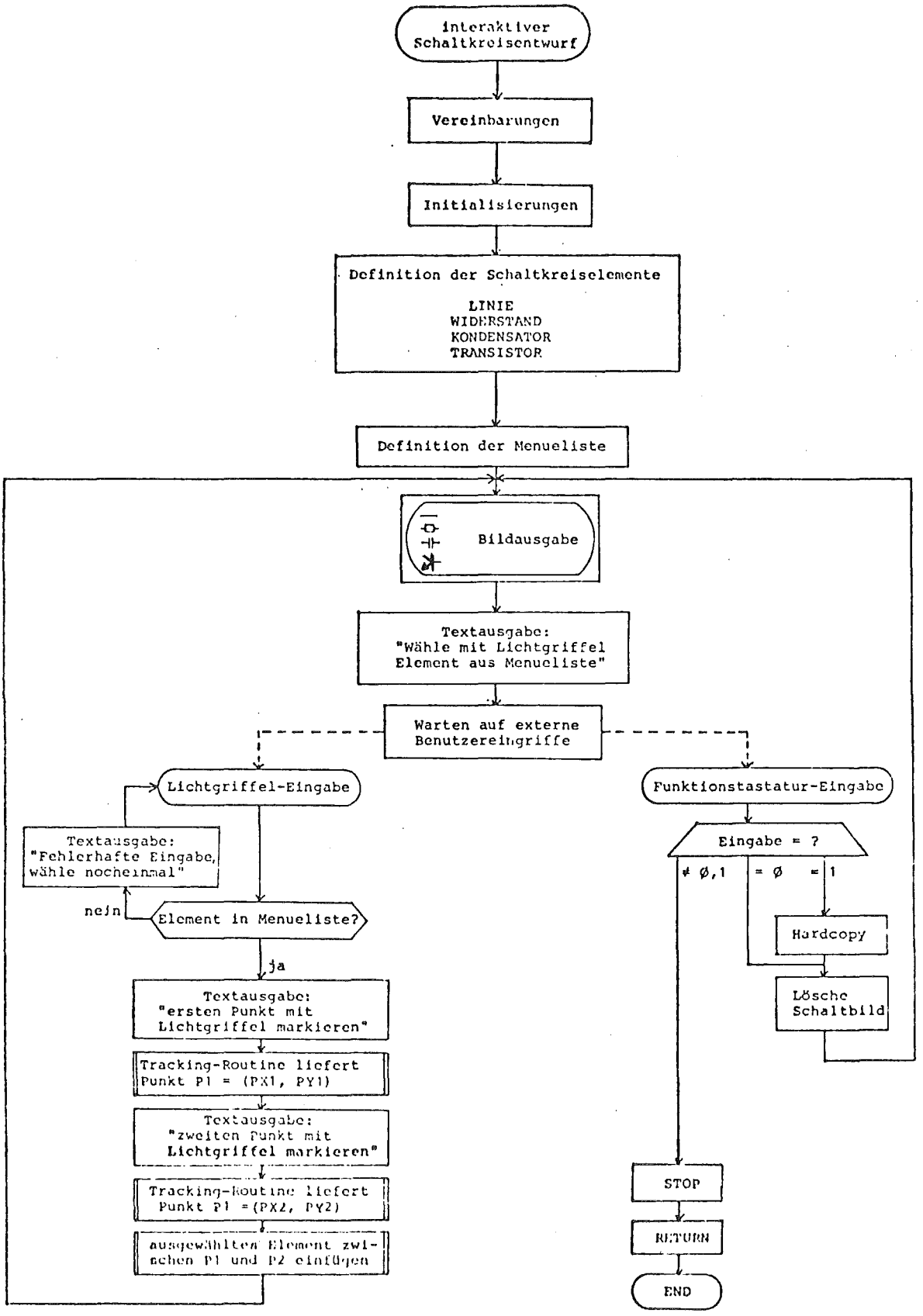


Bild 5-3: Ablaufdiagramm eines interaktiven Schaltkreisentwurfs

zu 1) Die Menüliste wird als Bild, bestehend aus den Teilbildern LINIE, WIDERSTAND, KONDENSATOR und TRANSISTOR, definiert. Auf diese Teilbilder kann mit einem Lichtgriffel gezeigt werden.

zu 2) Anschließend wird die Menüliste auf den Bildschirm ausgegeben, mit dem Text: "Wähle mit Lichtgriffel Element aus Menüliste". Das Programm wartet dann auf einen externen Benutzereingriff (Lichtgriffel oder Funktionstastatur).

zu 3) Der Benutzer zeigt mit dem Lichtgriffel auf ein gewünschtes Schaltelement in der Menüliste.
Hat er auf ein Element gezeigt, wird dieses zwischen zwei vom Benutzer zu markierenden Schirmkoordinaten P1 und P2 eingefügt, die er nach der Textausgabe "Ersten (bzw. zweiten) Punkt mit Lichtgriffel markieren" auf der Schirmfläche markieren kann. Eine Trackingroutine übergibt dem Programm die gewünschten Punktkoordinaten. Diese Routine wird als 'black box' behandelt, ebenso das Einfügen des ausgewählten Elements zwischen den beiden markierten Punkten.
Ein dem Schaltkreis hinzugefügtes Schaltelement behält seine ursprüngliche Größe (wie in der Menüliste).
Die dem Schaltkreis hinzugefügten Elemente WIDERSTAND, KONDENSATOR und TRANSISTOR behalten ihre Lage und Größe bei. Paßt eines dieser Elemente nicht zwischen P1 und P2, wird es so ausgegeben, daß der Endpunkt des Elements auf P2 liegt. Von P1 wird dann eine Verbindung zum Anfangspunkt des Elements gezogen.
Das Element LINIE wird in jedem Fall zwischen P1 und P2 eingepaßt.
Anschließend wird das hinzugefügte Schaltelement auf dem Bildschirm ausgegeben und auf eine erneute Benutzereingabe gewartet.

zu 4) Der Benutzer hat die Möglichkeit dreier Funktionstastatur-Eingaben. Gibt er die Funktionstaste 1 ein, so erhält er eine Hardcopy der aktuellen Bildschirminformation; anschließend wird das Schaltbild (nicht Menüliste!) auf der Schirmfläche gelöscht und auf eine erneute Benutzereingabe gewartet. Nach Eingabe der Funktionstaste Ø wird nur das Schaltbild gelöscht und auf eine neue Eingabe gewartet. Bei einer Eingabe einer Funktionstaste ungleich Ø und 1 wird das Programm beendet.

Das zugehörige Programm ist im Anhang aufgelistet.

6. Fehlermeldungen

Im Fehlerfall wird von LLGL die Task 'Fehler' aktiviert, die auf die Einheit mit der Nummer 5 eine Fehlermeldung ausgibt. Der den Fehler generierende Befehl wird nicht ausgeführt. Eine Fehlermeldung hat folgendes Aussehen:

Fehler <Fehlercode> in Task <Taskname>

Dabei ist <Taskname> der Name der Task, von der der fehlerhafte Befehl aus aufgerufen wurde.
<Fehlercode> ist einer der folgenden Fehlercodes:

Fehlercode	Bedeutung
1	keine Speicherplatzzuteilung möglich; Befehl wird nicht ausgeführt
2	mehr CLOSEs als OPENs
5	schon ein EXTEND offen oder unerlaubtes Setzen eines Attributes in einem durch EXTEND eröffneten Objekt
6	einem ENDEXT ging kein EXTEND voraus
201	es soll eine unbenannte Parameterliste generiert werden
202	ungültiger Objektname
203	Objektname ist noch nicht benutzt
204	ungültiger Objekttyp
205	Objekt wird zuoft referenziert
206	Parameterliste für ein Stringobjekt muß Koordinaten haben
207	das Objekt wird noch referenziert
208	das aktuell offene Objekt wird falsch referenziert
209	Objektname ist bereits belegt
210	ungültiger Parameterwert
211	Parameterliste ist zu lang
212	Objekt ist nicht erweiterbar
400	unzulässiger Eingabetyp

Fehler 4 und 200 verweisen auf LLGL-interne Fehler

Literatur

/1/ H. Grauer
LLGL - Eine Sprache für die interaktive graphische Programmierung
KFK 2414

/2/ H. Grauer, W. Müller, V. Jarsch
LLGL - Eine anwendungsinvariante interaktive graphische Sprache
(1976) unveröffentlicht

/3/ W. Müller
Ausgabeobjekte und zugehörige Operationen in LLGL
(1976) unveröffentlicht

/4/ H. Grauer
Die Eingabeverwaltung von LLGL
(1976) unveröffentlicht

/5/ ADAGE GP400 Users Reference Manual
ADAGE Inc., Boston

A n h a n g

Beispielprogramm-Listing

```
    1          COMMON /FUNKT/NUM
    2          CALL SCHALT
    3          STOP
    4          END
  0 ERRORS COMPILATION COMPLETE
```

```
      SUBROUTINE SCHALT
C
C          VEREINBARUNGEN/INITIALISIERUNGEN
C
      COMMON /FUNKT/ NUM
      EXTERNAL FUNTAS
      INTEGER OBJNAM(20)
C
      REAL XWID(9)/50.0,50.0,100.0,100.0,50.0,50.0,0.0,100.0,150.0/,
     1     YWID(9)/0.0,-25.0,-25.0,25.0,25.0,0.0,0.0,0.0,0.0/
      INTEGER HDWID(9)/0,1,1,1,1,1,1,0,1/
      REAL XKON(8)/0.0,70.0,70.0,70.0,80.0,80.0,80.0,150.0/,
     1     YKON(8)/0.0,0.0,25.0,-25.0,-25.0,25.0,0.0,0.0/
      INTEGER HDKON(8)/0,1,0,1,0,1,0,1/
      REAL XTRA(12)/0.0,75.0,75.0,150.0,75.0,75.0,75.0,100.0,90.0,
     1              100.0,100.0,150.0/,
     2     YTRA(12)/0.0,0.0,25.0,75.0,0.0,-25.0,0.0,-25.0,-25.0,
     3              -15.0,-25.0,-75.0/
      INTEGER HDTRA(12)/0,1,1,0,1,1,0,1,1,0,1,1/
C
      INTEGER TEXT1(26),TEXT3(22),TEXT4(22)
      CALL SETARR(TEXT1,
     1'WAEHLE  MIT  LICHTGRIFFEL  ELEMENT  AUS  MENUELISTE ',
     126)
      CALL SETARR(TEXT3,
     1'ERSTEN  PUNKT  MIT  LICHTGRIFFEL  MARKIEREN ',
     122)
      CALL SETARR(TEXT4,
     1'ZWEITEN  PUNKT  MIT  LICHTGRIFFEL  MARKIEREN!',
     122)
C
C
C          DEFINITION DES ATTENTION-HANDLING MIT DEM LICHTGRIFFEL
C          EINE EINGABE UEBER DIE FUNKTIONSTASTATUR WIRD
C          ASYNCHRON ABGEARBEITET (SIEHE ATTENTIONTASK).
C
      CALL GPON
      CALL ADEF(1,NUM,0,0)
      CALL ADEF(2,OBJNAM,0,0)
C
C          DEFINITION DER SCHALTKREISELEMENTE
C
C          LINIE (NAME=2)
      CALL UNITS(8,8,30000)
      CALL NAME(0,2)
      CALL START(1)
      CALL VEKTO2(0.0,0.0,1.0,150.0,0.0)
      CALL END
C
C          WIDERSTAND (NAME=3)
C
      CALL NAME(0,3)
      CALL START(1)
      CALL POLY2(XWID,YWID,1.0,HDWID,9)
      CALL END
C
C          KONDENSATOR (NAME=4)
C
      CALL NAME(0,4)
      CALL START(1)
      CALL POLY2(XKON,YKON,1.0,HDKON,8)
      CALL END
C
```

```
C          TRANSISTOR (NAME=5)
C
      CALL NAME(0,5)
      CALL START(1)
      CALL POLY2(XTRA,YTRA,1.0,HDTRA,12)
      CALL END
C        DEFINITION UND SOFORTIGE AUSGABE
C        DER MENUELISTE (NAME=6)
C
      CALL NAME(0,6)
      CALL START(0)
      CALL DISPLA(0,1,-1000.,0.,1.0)
      CALL REF(2)
      CALL DISPLA(0,1,-1000.,-200.,1.0)
      CALL REF(3)
      CALL DISPLA(0,1,-1000.,-400.,1.0)
      CALL REF(4)
      CALL DISPLA(0,1,-1000.,-600.,1.0)
      CALL REF(5)
      CALL END
      CALL DISPLA(0,0,0.,0.,1.)
C
C        DEFINITION UND SOFORTIGE AUSGABE
C        DES SCHALTKREISES (NAME=7)
C
    5 CONTINUE
      NUM=0
      CALL LIGHTP(0,-1)
      CALL SYMSCA(0,1,0.12)
      CALL NAME(0,7)
      CALL START(0)
      CALL SYMSCA(0,0,0.0)
      CALL LIGHTP(0,0)
   10 CALL NAME(0,8)
      CALL STRING(TEXT1,26,1,-1000.,-800.,1.)
      CALL ASTATE(1,FUNTAS)
      IF (NUM.EQ.-1) GOTO 5
C
C        WARTEN AUF LICHTGRIFFEL-EINGABE
C
 2000 CALL AWAIT(2)
      CALL DEL(8)
C
C        VERKNUEPFUNGSPUNKTE MARKIEREN
C
   40 CALL NAME(0,10)
      CALL STRING(TEXT3,22,1,-1000.,-800.,1.)
      CALL ASTATE(1,FUNTAS)
      IF (NUM.EQ.-1) GOTO 5
      CALL TRACK   (PX1,PY1)
      CALL DEL(10)
      CALL NAME(0,11)
      CALL STRING(TEXT4,22,1,-1000.,-800.,1.)
      CALL ASTATE(1,FUNTAS)
      IF (NUM.EQ.-1) GOTO 5
      CALL TRACK   (PX2,PY2)
      CALL DEL(11)
C
C        AUSGEWAEHLTES ELEMENT DEM SCHALTKREIS HINZUFUEGEN
C
      IF (OBJNAM(4).EQ.2) GOTO 50
      CALL EINF(OBJNAM(4)  ,PX1,PY1,PX2,PY2)
      GOTO 10
```

```
  127      50 CONTINUE
  128         CALL VEKTO2(PX1,PY1,1.0,PX2,PY2)
  129         GOTO 10
  130         END
  0 ERRORS COMPILATION COMPLETE
```

```
PAGE     1                          VORTXII    FTN IV(G)              0000 HOURS

     1          SUBROUTINE TRACK    (X,Y)
     2   C                 .
     3   C                 .
     4   C                 .
     5   C         UEBERGIBT DIE SCHIRMKOORDINATEN
     6   C         DER LICHTGRIFFELPOSITION
     7   C                 .
     8   C                 .
     9   C                 .
    10          RETURN
    11          END
  0 ERRORS COMPILATION COMPLETE
```

```
PAGE     1                          VORTXII    FTN IV(G)              0000 HOURS

     1          SUBROUTINE EINF(NAME,PX1,PY1,PX2,PY2)
     2   C                 .
     3   C                 .
     4   C                 .
     5   C         POSITIONIERUNG DES SCHALTKREISELEMENTES 'NAME'
     6   C         AN HAND DER VERKNUEPFUNGSPUNKTE (PX1,PY1)
     7   C         UND (PX2,PY2).
     8   C                 .
     9   C                 .
    10   C                 .
    11          CALL DISPLA(0,1,PX2-150,PY2,1.0)
    12          CALL REF(NAME)
    13          CALL DISPLA(0,0,0.,0.,0.)
    14          CALL VEKT02(PX1,PY1,1.0,PX2-150,PY2)
    15          RETURN
    16          END
  0 ERRORS COMPILATION COMPLETE
```

```
    1          SUB-   TINE FUNTAS
    2          COM-   /FUNKT/NUM
    3          IF (NUM-2) 20,10,30
    4       10 CALL HACO
    5       20 CONTINUE
    6          CALL END
    7          CALL DEL(7)
    8          NUM=-1
    9          RETURN
   10       30 CALL GPOFF
   11          STOP
   12          END
  0 ERRORS COMPILATION COMPLETE
```

PAGE 1 VORTXII FTN IV(G) 0001 HOURS

```
    1          SUBROUTINE SETARR(ARR,TEXT,LAENGE)
    2    C DIESES PROGRAMM INITIALISIERT EIN ARRAY MIT EINEM GEGEBENEN TEXT
    3    C
    4    C
    5          INTEGER ARR(1),TEXT(1),LAENGE
    6          DO 10 I=1,LAENGE
    7              ARR(I)=TEXT(I)
    8       10 CONTINUE
    9          RETURN
   10          END
  0 ERRORS COMPILATION COMPLETE
```