KERNFORSCHUNGSZENTRUM

KARLSRUHE

Dezember 1976 KFK 2414

Institut für Datenverarbeitung in der Technik

LLGL — Eine Sprache für die interaktive graphische Programmierung

H. Grauer

GESELLSCHAFT
FÜR
KERNFORSCHUNG M.B.H.

KARLSRUHE

GESELLSCHAFT FÜR KERNFORSCHUNG M. B. H.

KARLSRUHE

KERNFORSCHUNGSZENTRUM KARLSRUHE

KFK 2414

Institut für Datenverarbeitung in der Technik

LLGL - Eine Sprache für die interaktive graphische Programmierung

H. Grauer

Gesellschaft für Kernforschung m.b.H., Karlsruhe

Kurzfassung

LLGL (Low-Level-Graphical-Language) ist eine interaktive graphische Sprache mit ALGOL60-ähnlichen Bildstrukturierungseigenschaften, die z. Z. in Form eines von FORTRAN aus aufrufbaren Unterprogrammpaketes implementiert ist. Die Sprache erlaubt die Definition und Manipulation von hierarchisch gegliederten Mengen zwei- oder dreidimensionaler graphischer Objekte und zugeordneten graphischen Attributen sowie graphische Interaktionen des Benutzers. Sie ist besonders zur Programmierung von interaktiven graphischen Problemen wie sie im CAD vorliegen, geeignet.
Nach einem einführenden Überblick über allgemeine Charakteristika graphischer Sprachen wird das Sprachkonzept von LLGL ausführlich dargestellt und der Befehlsumfang beschrieben. Abschließend werden an einem Programmbeispiel die wesentlichen Spracheigenschaften zusammenfassend gezeigt.

LLGL - A Language for interactive graphical programming

Abstract

LLGL (Low-Level-Graphical-Language) is an interactive graphical language with ALGOL60-like features for image-structuring.
It is implemented at the moment as a FORTRAN callable subroutine package. The language permits the definition and manipulation of hierarchical ordered sets of two- or three-dimensional graphical objects with graphical attributes as well as interactions of the user. It is specifically suitable for programming of interactive graphical CAD-problems.
After an introductory overview of general characteristics of graphical languages the language concept of LLGL is described in detail and the set of statements shortly depicted. Finally, the essential features of the language are exemplified in a program.

Der vorliegende Bericht ist das Manuskript eines Vortrages, welcher am 11.05.76 anläßlich eines Seminars über Computer-Graphik im Institut für Graphische Datenverarbeitung und Strukturerkennung (IGS) der Gesellschaft für Mathematik und Datenverarbeitung (GMD) gehalten wurde.

Inhalt

1. Einleitung

Die Computer-Graphik hat sich in den letzten Jahren zu einem sehr wichtigen Hilfsmittel der Mensch-Rechner-Kommunikation, besonders in dem Bereich Entwurf und Konstruktion, entwickelt. Das liegt an der dominierenden Rolle, die die zweidimensionale Information auf dem Gebiet des technologischen Entwurfs spielt. Hier sind Bilder in Form von Linienzeichnungen, wie Diagramme, Fließbilder, Konstruktionspläne etc. schon immer ein wesentliches Informationsmedium des Konstrukteurs und Ingenieurs gewesen.

Die beliebig oft reproduzierbare Erzeugung dieser Zeichnungen durch den Rechner, die Fähigkeit des Rechners, die Daten der Zeichnung mit zugehörigen Problemdaten zu verbinden und in Algorithmen direkt zu verarbeiten sowie die Möglichkeit der Archivierung der erzeugten graphischen Daten zusammen mit den Problemdaten und damit die Weitergabe an nachfolgende Stufen des Entwurfs- sowie des Produktionsprozesses machen den Rechner, in Verbindung mit der Computer-Graphik, hier zu einem unerhört nützlichen und leistungsstarken Werkzeug.

Das rechnergestützte Entwickeln und Konstruieren (engl. CAD = Computer Aided Design) profitiert insbesondere auch von den interaktiven Möglichkeiten der Computer-Graphik. Gerade im Entwurfsprozeß besteht der typische Arbeitsablauf zum großen Teil darin, graphische Informationen zu erzeugen und dauernd zu manipulieren.

Elementare Voraussetzung, damit die Computer-Graphik die hier an sie gestellten Anforderungen erfüllen kann, ist eine geeignete graphische Hardware, also Sichtgeräte für die dynamische graphische Ausgabe zusammen mit Plottern für die Erzeugung von statischen Zeichnungen und Eingabegeräte wie Lichtgriffel oder Cursor sowie Tastaturen für Interaktionen des Benutzers.

Was allerdings oft übersehen wird ist, daß erst eine leistungsfähige graphische Software die problemangepaßte und komfortable Nutzung der Möglichkeiten der Computer-Graphik schafft.

Die zum Betrieb eines graphischen Systems nötige Software wird als graphische Programmiersprache bezeichnet. Die Motivation mit LLGL (Low-Level Graphical-Language) den schon existierenden graphischen Programmsprachen eine weitere hinzuzufügen lag darin begründet, daß die existierenden und verfügbaren graphischen Sprachen den hier [1] gestellten Anforderungen nicht oder nur teilweise genügten. Die wesentlichen Anforderungen waren:

- Wohlstrukturierte und leicht verständliche Sprachdefinition
- Leistungsfähige und einfach zu handhabende Sprachmittel zur interaktiven Manipulation von Bildern
- Einsatz auf Kleinrechnern in einer Multi-Tasking-Umgebung
- Effektive Nutzung von leistungsstarker Sichtgerätehardware

Bevor die Spracheigenschaften von LLGL im einzelnen beschrieben und mit einem Beispiel verdeutlicht werden, soll zunächst als Einführung in die Thematik ein Überblick über allgemeine Charakteristika graphischer Sprachen gegeben werden.

---

[1] LLGL wurde am Institut für Datenverarbeitung in der Technik der Gesellschaft für Kernforschung für eine VARIAN V75 - ADAGE GP400 Rechnersichtgerätekonfiguration entwickelt.

2. Spracheigenschaften und Implementierungsformen graphischer Sprachen

Was ist eine graphische Sprache?

Graphische Sprachen sind eine Klasse der Programmiersprachen, die wiederum eine Klasse der künstlichen Sprachen sind.

Sie zeichnen sich durch ihren besonderen Objektbereich aus, nämlich Bilder, die einem bestimmten Problembereich zugeordnet sind. Das bedeutet also auch: die Bilder sind kein Selbstzweck, sondern dienen dazu, ein Problem zu lösen.

Als graphische Sprache soll im weiteren eine Sprache verstanden werden, die als Eingabe eine Bildbeschreibung verarbeitet und als Ausgabe ein Bild erzeugt, wobei hier nur Bilder, die aus Linien bestehen, betrachtet werden sollen (Liniengraphik).
Wenn diese Bildbeschreibung während des Ablaufs eines graphischen Programms durch Eingaben des Benutzers geändert werden kann (Dialog), spricht man von einer interaktiven graphischen Sprache.

(1.) Graphische Grundobjekte (Primitive)

Jede graphische Sprache muß über graphische Grundobjekte verfügen wie Vektoren (hell, dunkel, absolut, relativ) und alphanumerische Zeichen. Ein wesentlicher Unterschied besteht hier darin, ob diese Grundobjekte nur in einem zwei-dimensionalen oder auch in einem 3-dimensionalen Koordinatensystem gezeichnet werden können.

Weitergehende Fähigkeiten - deren Nutzen stark anwendungsabhängig ist - sind Grundobjekte wie Kreise, Kreisbögen oder allgemeinere Kurven, Flächen und Körper.

(2.) Bildstrukturierung

Bildstrukturierung bedeutet, daß den graphischen Grundobjekten eine Struktur aufgeprägt werden kann, die es erlaubt, das Bild nicht nur als ungeordnete Menge von Punkten oder Linien zu sehen. Die Struktur des Bildes sollte die Struktur des dargestellten Problems widerspiegeln.

Mögliche Bildstrukturformen sind:

(a) Segmente
Ein Segment ist eine Kollektion von graphischen Grundobjekten. Ein Bild kann in beliebig viele Segmente unterteilt werden. Dieses ist die einfachste Form der Bildstrukturierung.

(b) Baum
Die graphischen Grundobjekte sind die Blätter des Baumes. Die übrigen Knoten tragen Attribut- und Strunkturinformation. Mehrmalige Verwendung von gleichen Bildkomponenten ist nur durch Erstellen von Kopien möglich.

(c) Gerichteter Graph
Zusätzlich zu den Eigenschaften von (b) ist die mehrmalige Verwendung von gleichen Bildkomponenten möglich.

(3.) Änderung der Bildstruktur

Für eine interaktive Arbeitsweise ist es wichtig, die Bildstruktur durch Operationen wie Hinzufügen, Löschen oder Umordnen von Komponenten ständig ändern zu können.

(4.) Attribute

Attribute kontrollieren Eigenschaften von graphischen Objekten wie
- Selektionsattribute (Namen, Lichtgriffelempfindlichkeit)
  Selektionsattribute sind Voraussetzung für (2.) und (3.)
- Textur (Linienart, Farbe, Blinken etc.)
- Topologie (Transformationen, Schreibrichtung etc.)

Es ist vorteilhaft, wenn Attribute an jeder Stelle der Bildstruktur - also sowohl zu graphischen Grundobjekten als auch zu in der Struktur höher stehenden Objekten - und zu jeder Zeit gesetzt werden können.

(5.) Interaktive Fähigkeiten

Hierunter sollen Sprachmittel verstanden werden, die eine Interaktion des Benutzers mit seinem Anwendungsprogramm vom Bildschirmarbeitsplatz aus erlauben. Die Interaktion kann "nichtgraphischer" Natur sein, z.B. Eingabe von Zeichenstrings (Kommandos, Objektnamen) über eine alphanumerische Tastatur oder "graphischer" Natur, z.B. Zeigen auf ein Objekt auf dem Bildschirm.

Die dafür zur Verfügung stehenden Sprachmittel können von einfacher tabellarischer Spezifikation der Eingaben und synchroner Verarbeitung bis hin zur asynchronen Verarbeitung mit Eingabeprozeduren (ON - Feature in PL1) reichen.

(6.) Fähigkeiten zum Aufbau von dynamischen Datenstrukturen

Die Möglichkeit, Daten mit den Mitteln der Sprache einfach zu strukturieren und zu verknüpfen, ist sowohl bei den graphischen Daten als auch bei den problembezogenen Daten eines graphischen Systems eine wertvolle Untersützung, um die Beziehungen der graphischen Daten und Problemdaten unter- und zueinander darstellen zu können.

(7.) Fähigkeiten zur Problemlösung:Algorithmenformulierung und Programmkontrolle

Man darf nicht vergessen, daß bei vielen Anwendungen der graphische Programmteil gegenüber den umfangreichen Berechnungen, die mit den erzeugten graphischen Daten durchgeführt werden,

relativ klein ist. Deswegen sind die Sprachmittel zur Algorithmenformulierung wie arithmetische und logische Befehle sowie flexible und benutzungsfreundliche Möglichkeiten zur Programmkontrolle von großer Wichtigkeit.

Wie realisiert man eine graphische Sprache?

Es gibt verschiedene Möglichkeiten zur Implementierung einer graphischen Sprache:

(1.) Eigenständige graphische Sprache

Die Entwicklung einer eigenständigen graphischen Sprache bietet sicher die günstigsten Voraussetzungen zur Verwirklichung der oben genannten Spracheigenschaften, da hierbei theoretisch keine einschränkenden Randbedingungen beachtet werden müssen.

Der Aufwand für die Realisierung ist jedoch sehr groß; es müssen zudem viele Eigenschaften dupliziert werden, die in gebräuchlichen höheren Programmiersprachen bereits vorhanden sind. Kritische Punkte sind Verträglichkeit mit bereits vorhandenen Anwenderprogrammen in anderen Sprachen, Verbreitungsmöglichkeiten und Übersetzerwartung.

Übersetzererzeugende Systeme, die zur Zeit noch von rein akademischem Interesse sind, könnten hier in Zukunft zu einer befriedigenden Lösung führen.

(2.) Integration in eine vorhandene höhere Programmiersprache

Die Grammatik der Gastgebersprache sei G_{gast} die Grammatik des zu integrierenden graphischen Sprachanteils sei G_{graph} Dann gilt:

$$L_G = L\ (G_{gast} + G_{graph})$$

Die resultierende Sprache L_G ist die von der Vereinigung der Grammatiken beider Sprachanteile erzeugte Sprache.
Wie kann die Integration des graphischen Sprachanteils erfolgen?

(a) Syntaxerweiterung der Gastgebersprache
Aus Gründen der Übertragbarkeit und des Implementierungsaufwandes ist diese Vorgehensweise nur dann günstig, wenn die Gastgebersprache per definitionem eine erweiterbare Sprache ist.

(b) Vorübersetzer
Die Vorübersetzung des graphischen Sprachanteils in Befehle der Gastgebersprache ist sowohl was die Lesbarkeit der Sprache als auch den Implementierungsaufwand angeht, ein sinnvoller Ansatz, der inzwischen in einer Anzahl von Implementierungen verwirklicht wurde.

(c) Graphisches Unterprogrammpaket
Die Integration des graphischen Sprachanteils in Form einer Unterprogrammbibliothek in die Gastgebersprache ist die einfachste und flexibelste Implementierungsform. Die Übertragbarkeit bereitet keine Probleme; auch kann die Bibliothek Schnittstellen zu verschiedenen Gastgebersprachen enthalten, da fast alle höheren Programmiersprachen den Unterprogrammaufruf als Sprachelement kennen. Diesen Vorteilen steht allerdings als Nachteil entgegen, daß die Erstellung und Lesbarkeit der graphischen Programme stark darunter leiden, daß die Syntax und Semantik des graphischen Sprachanteils nur mittels der Namen der Unterprogramme und den zugehörigen Parameterlisten ausgedrückt werden können.

Geeignete Gastgebersprachen für die Integration eines graphischen Sprachanteils sind: ALGOL68 - allerdings wegen fehlender Implementierungen nur theoretisch, PASCAL, SIMULA, die bis jetzt noch keine Beachtung gefunden haben, PL1, APL für die mehrere graphische Erweiterungen existieren. FORTRAN ist sicherlich von seinen Spracheigenschaften her gesehen am wenigsten geeignet, hat jedoch den Vorteil, praktisch auf allen Rechnern standardisiert verfügbar zu sein und wurde deswegen am häufigsten als Gastgebersprache benutzt.

3. Die Eigenschaften der interaktiven graphischen Sprache LLGL

3.1 Allgemeines

LLGL ist eine graphische Spracherweiterung in Form einer Unterprogrammbibliothek. Es ist zur Zeit von FORTRAN aus benutzbar. Schnittstellen zu anderen Sprachen sind prinzipiell leicht möglich. LLGL ist auf einem Kleinrechner implementiert und in einer Multi-tasking Umgebung ablauffähig. Dabei können LLGL-Befehle in beliebigen parallel ablaufenden Tasks benutzt werden. Alle solche "graphischen" Tasks arbeiten auf ein gemeinsames Bild und kommunizieren bezüglich der Graphik über die allen Tasks gemeinsamen Namen der graphischen Objekte in diesem Bild. Damit wird der modulare Aufbau eines graphischen Anwenderprogrammsystems stark unterstützt. Gleichzeitig ist dadurch die Größe des graphischen Programmteils nicht durch den logischen Adreßraum (bei Kleinrechnern beschränkt auf 64 K Byte) sondern nur durch den physikalischen Speicherausbau (bei Kleinrechnern z. Z. bis 512 K Byte möglich) beschränkt.

3.2 Graphische Grundobjekte und Bildstrukturierung

LLGL ermöglicht den Aufbau von Bildstrukturen in Form eines gerichteten schleifenfreien Graphen. Im einzelnen stehen für den Aufbau eines Bildes folgende Elemente zur Verfügung:

Primitive Objekte

- Vektor: Dieses Objekt erzeugt einen Vektor (2D/3D)
- Polygon: Dieses Objekt erzeugt einen Linienzug, der eine beliebige Anzahl von Stützpunkten verbindet (2D/3D)
- Symbolstring: Mit diesem Objekt wird eine Zeichenkette ausgegeben.

- Referenz: Dieses Objekt referenziert ein höheres, referenzierbares Objekt (s.u.). Das höhere Objekt wird dadurch sichtbar gemacht.

Parameterlisten

Für die primitiven Objekte Vektor, Polygon und Symbolstring können Parameterlisten (Felder von Typ Vektor, Polygon, String) definiert werden, die von beliebig vielen dieser Objekte als Wertebereich benutzt werden können.

Höhere Objekte

Primitive Objekte können zu höheren Objekten zusammengefaßt werden, welche wiederum, zusammen mit primitiven Objekten zu höheren Objekten zusammengefaßt werden können. Man unterscheidet bei dieser Objektart zwischen dem referenzierbaren und dem nicht referenzierbaren, höheren Objekt.

- nicht referenzierbares, höheres Objekt

Das nicht referenzierbare, höhere Objekt dient der Zusammenfassung mehrerer Objekte (Mengenbildung), um z.B. Operationen oder Gültigkeitsbereiche von Attributen auf eine Menge von Objekten beziehen zu können. Es ist vergleichbar mit dem Block in höheren Programmiersprachen.

- referenzierbares, höheres Objekt

Das referenzierbare, höhere Objekt hat zusätzlich eine Funktion, die mit dem Unterprogramm in einer Programmiersprache vergleichbar ist. Dieses Objekt wird als Schablone einmal definiert und kann dann mittels des Objektes Referenz (s.o.) an beliebig vielen Stellen aufgerufen, d.h. sichtbar gemacht werden.

3.3 Operationen auf graphischen Objekten

Die in LLGL möglichen Operationen auf graphischen Objekten können in zwei Klassen aufgeteilt werden.

3.3.1 Operationen zum Aufbau und zur Änderung der Bildstruktur

An Operationen, die auf die Bildstruktur wirken, sind zu nennen:

(1.) Generierung von graphischen Objekten

Hierunter fallen alle Operationen, die primitive und höhere graphische Objekte generieren.
Die Generierung primitiver graphischer Objekte erfolgt durch Angabe des entsprechenden Objektbefehls, z.B. POLY3A (.....) zur Generierung eines Objektes drei-dimensionales Polygon. Die Generierung höherer graphischer Objekte erfolgt durch Setzen einer 'Klammer' mit START(art), END um die zu dem höheren graphischen Objekt gehörenden Objekte, wobei 'art' angibt, ob es sich um ein referenzierbares oder nicht referenzierbares höheres graphisches Objekt handelt.

(2.) Löschen von graphischen Objekten

Alle benannten (siehe 3.3.2) graphischen Objekte können durch die Löschoperation DELETE(name) iweder aus der Bildstruktur entfernt werden.

(3.) Erweitern von graphischen Objekten

Durch die Erweiterungsoperation können in eine Bildstruktur an beliebiger Stelle neue graphische Objekte, also neue graphische Teilstrukturen, eingefügt werden. Ein zu erweiterndes graphisches Objekt wird mit dem Befehl EXTEND(name) wieder 'geöffnet', dann folgt die Generierung der hinzuzufügenden Objekte nach (1.).
Mit ENDEXT wird die Erweiterung abgeschlossen.

3.3.2 Operationen, die Attribute von graphischen Objekten setzen

Mit diesen Operationen können für graphische Objekte zu jeder Zeit Attribute gesetzt bzw. rückgesetzt werden. LLGL kennt die folgenden Attribute:

(1.) Attribute für die Selektion von graphischen Objekten

- NAME (old, new)
  Mit dem Attribut NAME kann einem graphischen Objekt ein Name (ganze Zahl) gegeben werden bzw. ein Name geändert werden. Zwei Namen haben eine Sonderbedeutung:
  old = Ø bezeichnet das nächste zu generierende Objekt,
  old = 1 bezeichnet das augenblicklich generierte Objekt.
  Mit Hilfe dieser Sondernamen können neuen Objekten Namen gegeben werden.

  Bei allen des weiteren aufgeführten Attributen dient der Name zur Kennzeichnung des graphischen Objekts, für welches die Attributfunktion ausgeführt werden soll.
  Zu erwähnen ist hier noch eine besondere Eigenschaft von LLGL: Beim Zeigen auf ein unbenanntes graphisches Objekt mit dem Lichtgriffel auf dem Bildschirm generiert LLGL einen Systemnamen. Dieser Systemname kann im weiteren vom Benutzer wie ein von ihm selbst generierter Name verwendet werden - er kann also z.B. durch Zuweisung an die Größe 'old' zur Generierung eines Benutzernamens für dieses Objekt verwandt werden.

- LIGHTP (name,....)
  Mit diesem Attribut kann die Selektionsempfindlichkeit eines graphischen Objekts für den Lichtgriffel gesetzt werden.

(2.) Attribute, die das Aussehen von Linien oder Symbolen ändern

- BLINK (name,...)
  Für Blinken von Vektoren oder Symbolen.
- VECTEX (name,...)
- SYMTEX (name,...)
  Für die Textur von Vektoren bzw. Symbolen
- UNDERL (name,...)
  Zur Unterstreichung von Symbolen

(3.) Attribute, die die Topologie eines graphischen Objektes beeinflussen

Für Vektoren:
Hierunter fallen die graphischen Transformationen Verschiebung, Skalierung und Rotation.

- DISPLA (name,...)
- VECSCA (name,...)
- ROTA (name,...)

Für Zeichenketten:
- SYMSCA (name,...)
- SYMDIR (name,...)
- SYMREF (name,...)

Zur Skalierung, Schreibrichtungsbestimmung und Drehung von Symbolen.

Neben diesen Attributen, die gezielt auf einzelne graphische Objekte wirken, gibt es noch drei Attribute, die auf ein ganzes Bild wirken, nämlich

- UNITS
- VIEWPORT
- VISUAL

zur Festlegung der Abbildung des Benutzerbildraumes auf den physikalischen Bildraum.

3.4 Interaktive Eigenschaften

Die interaktiven Eigenschaften einer graphischen Sprache sind zum einen durch die Sprachmittel für die Verarbeitung von Eingaben und zum anderen durch die Sprachmittel für die Änderung der Bildstruktur und der Attribute der graphischen Objekte bestimmt. So kann z.B. eine Interaktion darin bestehen, durch Zeigen mit dem Lichtgriffel auf ein graphisches Objekt den Namen dieses Objektes zu ermitteln (Eingabe) und mit Hilfe des Namens das Objekt zu löschen (Änderung der Bildstruktur) oder das Objekt blinken zu lassen (Änderung eines Attributs).

Die Sprachmittel von LLGL für die Bildstrukturmanipulation und die Attribute von graphischen Objekten wurden bereits ausführlich behandelt. Hier sollen nun die Sprachmittel für die Eingabe vorgestellt werden.

3.4.1 Eingabegeräte

LLGL unterstützt zur Zeit folgende Eingabegeräte:

a) alphanumerische Tastatur
b) Funktionstastatur
c) Lichtgriffel

Die Menge der Eingabegeräte ist erweiterbar, z.B. um Daten-Tablett, Potentiometer etc. Daneben gibt es natürlich die Eingabegeräte des Rechners, die ohnehin vom Betriebssystem unterstützt werden.

Die alphanumerische Tastatur dient zur Übergabe von Zeichenketten, die Funktionstastatur liefert Steuerinformation und mit dem Lichtgriffel können graphische Objekte auf dem Bildschirm identifiziert werden. Damit verfügt der Benutzer von LLGL über ein Spektrum von

Eingabegeräten, welches ihm den flexiblen Aufbau von interaktiven graphischen Programmen für eine große Klasse von Anwendungsproblemen ermöglicht.

3.4.2 Sprachmittel für die Eingabeverarbeitung

Die Verarbeitung von Eingaben mit LLGL setzt die Definition der Art der Eingabebehandlung voraus. In dieser Definition wird das gewünschte Eingabegerät dem Benutzerprogramm - genauer ausgedrückt der Benutzertask - zugeordnet und ein Datenbereich angegeben, in dem die Eingabedaten beim Eintreffen eines Eingabeereignisses abgelegt werden sollen.
Des weiteren kann der Eingabemodus bestimmt werden. Mit diesem kann man die Eingabe abhängig vom Wert der Eingabevariablen machen (bedingte Eingabe) und eine bestimmte Arbeitsweise des Eingabegerätes festlegen, z.B. beim Lichtgriffel zwischen der Betriebsart 'Identifizieren' oder 'Verfolgung' eines Cursorsymbols wählen.

Nach der Definition der Eingabebehandlung kann die Eingabeverarbeitung erfolgen. Hierfür gibt es in LLGL vom Programmkontrollfluß her gesehen drei Möglichkeiten:

(a) Abfragen auf Vorhandensein eines Eingabeereignisses:
ASTATE (gerät)
Wenn kein Eingabeereignis stattgefunden hat, läuft das Programm mit dem nächsten Befehl weiter, sonst führt es ein in den Eingabebefehl angegebenes Unterprogramm aus.

(b) Warten auf eine Eingabe: AWAIT (gerät)
Das Programm wartet mit einem Lesebefehl - unter Freigabe der CPU - bis eine Eingabe bei dem betreffenden Gerät vorliegt und läuft dann weiter.

(c) Asynchrone Behandlung der Eingabe
Da LLGL die Zuordnung von Eingabegeräten zu beliebigen Tasks erlaubt, kann der Benutzer zur asynchronen Behandlung der Eingabe in einem Multi-Tasking System eine oder mehrere Eingabegeräte einer oder mehreren eingabeverarbeitenden Tasks zuordnen und die betreffenden Eingaben dann in dieser Task abarbeiten (mit AWAIT). Die eingabeverarbeitenden Tasks laufen asynchron zur Hauptprogrammtask und die Abarbeitungsfolge wird nur durch die vom Benutzer vergebenen Prioritäten der Tasks bestimmt.

3.5 Strukturierung und Verknüpfung von graphischen Daten- und Problemdaten

Als graphische Daten sollen alle die Daten verstanden werden, die zur Darstellung auf dem Bildschirm dienen. Problemdaten sind die Daten, welche nichtgraphische Eigenschaften wie z.B. Material, Festigkeit, Preis etc. enthalten.

LLGL enthält keine Fähigkeiten, um für graphische Daten oder Problemdaten explizite Datenstrukturen aufzubauen. In vielen Anwendungsfällen kann der Benutzer auf den Aufbau einer expliziten Struktur für die graphischen Daten verzichten, da das von LLGL erzeugte Bild eine Struktur enthält, auf die für Bildmanipulationen zugegriffen werden kann (cf 3.2). Für die Archivierung von graphischen Daten sowie für die Verwaltung der Problemdaten müssen hingegen vom Benutzer Datenstrukturen explizit mit den Sprachmitteln der Gastgebersprache aufgebaut werden - welche zugegebenermaßen im aktuellen Fall für FORTRAN unbefriedigend sind. Bei dem Entwurf von LLGL wurde aber bewußt eine Konzentration auf die primär graphischen Fähigkeiten vorgenommen und nicht der Versucht gemacht, vorhandene Mängel der Gastgebersprache auszugleichen.

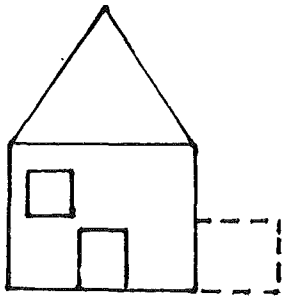
Hingegen wird die Verknüpfung von graphischen Daten und Problemdaten von LLGL durch die Möglichkeit stark unterstützt, jedes graphische Objekt mit einem Namen versehen zu können. Der Name eines graphischen Objekts kann als Schlüssel zum Auffinden der

zugehörigen Problemdaten dienen und umgekehrt kann über den zu den Problemdaten gehörenden Schlüssel jederzeit das zugehörige graphische Objekt auf dem Bildschirm angesprochen werden. So kann zum Beispiel der Name eines graphischen Objekts, welcher in LLGL durch eine ganze Zahl repräsentiert ist, als Index in einem Feld dienen, welches an dieser Stelle die Problemdaten selbst oder einen Verweis auf die Problemdaten enthält.

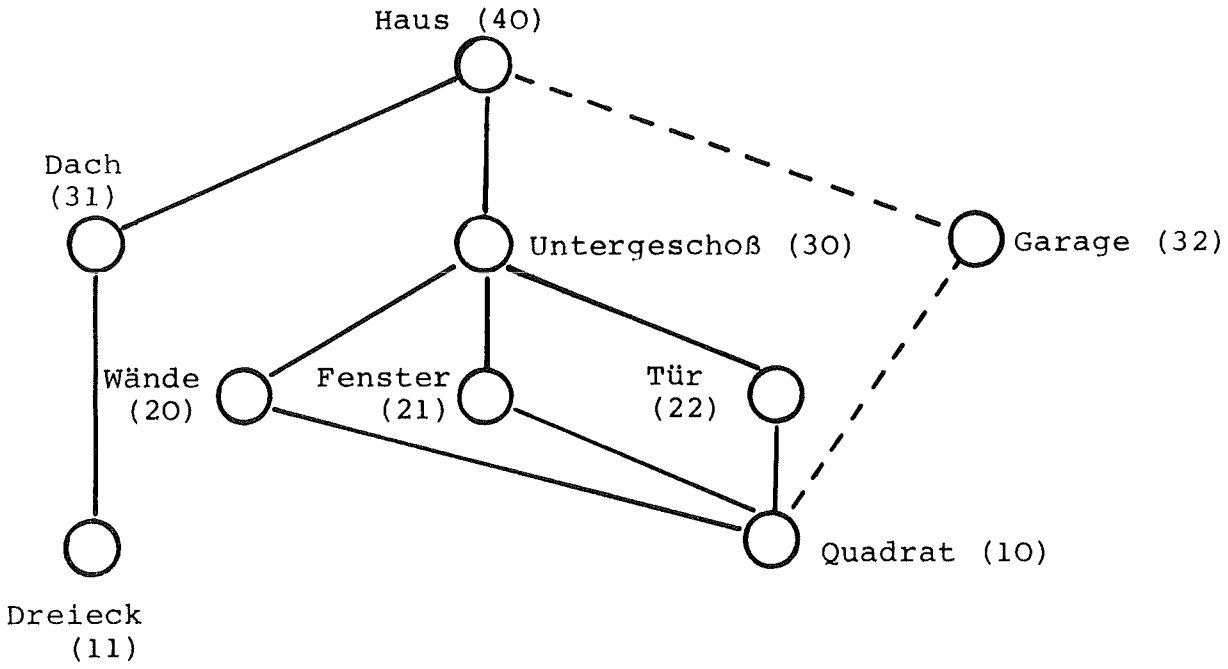
4. Die Programmierung mit LLGL

Ein einfaches Programmierbeispiel, welches als exemplarischer Ausschnitt aus einer echten Problemlösung angesehen werden kann, soll die Leistungsfähigkeit und leichte Handhabbarkeit von LLGL beim Aufbau von Bildstrukturen und der interaktiven Manipulation von Bildern zeigen. Die Verknüpfung mit der zugehörigen Problemdatenstruktur wurde hierbei der Übersichtlichkeit halber außer acht gelassen. Wie bereits gezeigt wurde, kann diese aber leicht mit Hilfe der Namen der graphischen Objekte hergestellt werden.

Es soll das folgende Bild aufgebaut werden:



Die Bildstruktur kann in Form eines gerichteten Graphen dargestellt werden.

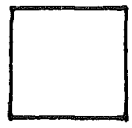
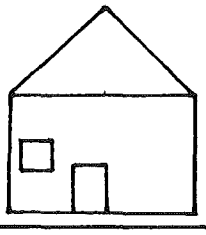


Die Objekte Quadrat und Dreieck sollen als Schablonen definiert werden, um mehrfach verwendet werden zu können. Alle Objekte sollen benannt sein, um sie identifizieren zu können.

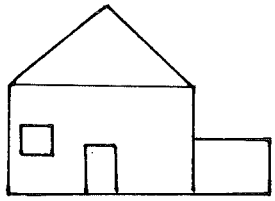
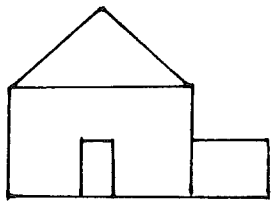
Das im Folgenden aufgeführte LLGL-Programm beschreibt den Aufbau des Bildes:[1)]

---

1) Der Übersichtlichkeit halber wurden in dem Programm das 'Call' vor jedem LLGL-Befehl weggelassen, sowie irrelevante Parameter nicht spezifiziert.

Bild	Befehle	Aktion
	Name (0,10) Start (1) Polya2 (Q,...) End	Definition einer Schablone Quadrat durch ein referenzierbares höheres Objekt mit dem Namen 10. Das Feld Q enthält die Koordinaten der Eckpunkte.
	Name (0,11) Start (1) Polya2 (D,...) End	Definition einer Schablone Dreieck (Name = 11)
	Name (0,40) Start (0)	Generierung des Objektes Haus (Name = 40)
	Name (0,30) Start (0)	Generierung des Objektes Untergeschoss (Name = 30)
	Name (0,20) Displa (0,...) Start (0) Vecsca (1,.) Ref (10) End	Generierung des Objektes Wände (Name = 20) und Zeichnen der Wände
	Name (0,21) Displa (0,...) Start (0) Vecsca (1,.) Ref (10) End	Generierung des Objektes Fenster (Name = 21) und Zeichnen des Fensters
	Name (0,22) Displa (0,...) Start (0) Vecsca (1,.) Ref (10) End	Generierung des Objektes Tür (Name = 22) und Zeichnen der Tür
	End	Abschluß des Objektes Untergesch.
	Name (0,31) Displa (0,...) Start (0) Vecsca (1,.) Ref (10) End	Generierung des Objektes Dach (Name = 31) und Zeichnen des Daches
	End	Abschluß des Objektes Haus

Das Bild soll nun durch Hinzufügen eines Objektes 'Garage' zu dem Objekt Haus und Löschen des Objekts 'Fenster' geändert werden. Anschließend soll das Objekt 'Garage' blinken.

Bild	Befehle	Aktion
	Extend (40)	Das Objekt Haus wird zur Erweiterung wieder geöffnet
	Name (0,32) Displa (0,..) Start (0,..) Vecsca (1,.) Ref (10) End	Generierung des Objektes Garage (Name = 32) und Zeichen der Garage
	Endext	Abschließen der Erweiterung
	Delete (20)	Löschen des Objektes Fenster
	Blink (32,1)	Blinken des Objektes Garage

Dieses Programm soll nun durch Hinzunahme von graphischer Eingabe zu einem interaktiven Programm erweitert werden.
So kann z.B. das Löschen von Objekten durch Zeigen mit dem Lichtgriffel auf das zu löschende Objekt gesteuert werden. Als zusätzliche Eingabe wird dabei die Angabe benötigt, welche Hierarchiestufe in dem gezeigten Objekt gemeint ist, d.h. ob z.B. beim Zeigen auf das Fenster dieses, das Untergeschoß oder das ganze Haus gelöscht werden soll. Diese Eingabe soll über die Funktionstastatur erfolgen.

Befehle	Aktion
Lightp (40,1)	Das Objekt Haus für den Lichtgriffel empfindlich machen
Adef (2, Namen,...) Adef (1, Stufe,...)	Definition von Lichtgriffel- und Funktionstastatureingabe. Namenshierarchie des gezeigten Objekts soll in aufsteigender Folge im Feld 'Namen', Hierarchiestufe in der Variablen 'Stufe' abgelegt werden.
Await (2) Await (1)	Warten auf Lichtgriffeleingabe Warten auf Funktionstastatureingabe
Delete (Namen (Stufe))	Löschen des gezeigten Objekts. Namen (Stufe) liefert den Namen des zu löschenden Objekts

Literatur

H. Grauer, R. Merkel
Interaktive graphische Systeme:
Allgemeine Charakteristika und Leistungskenngrößen einiger ausgewählter Systeme
KFK 2072, Oktober 1974

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

P. Kottmann
Graphische Sprachen - Übersicht, Klassifizierung und Bewertung
Diplomarbeit Universität Karlsruhe, Februar 1976

W. Müller, H. Grauer
Einführung in die graphische Programmierung mit LLGL
KFK 2415, Dezember 1976