HIGPSL：LISPの機能を活用したインタラクティブ・グラフィックス用の高水準プロセッシング・システム

間野暢興　古川康一
（電子技術総合研究所）

グラフィック・ディスプレイ装置を用いて、図形を対象として、あるいは図形という視覚に直接訴える媒体を介して、人間と計算機が対話を行なうシステムは最近かなり普及してきた。しかし、グラフィックスによるインタラクションの場面はマン・マシン・システムのもっとも複雑かつデリケートな場面であり，人間の意図を画面を通して正しくかつ能率的に計算機内のデータ構造に反映させることが必要なため、ひとつのアプリケーション・システムを完成させるだけでも、プログラミングとデバッギングにかなりの時間と労力を費しているのが現状である。これには、Graphic Subroutine PacKage（GSP）が低レベルであること、GSPとデータ構造管理ルーチンが別々になっていること、なども原因としてあげられる。

このインタラクティブ・グラフィックスのシステム作りの問題を統一的に解決するため、我々は高水準のインタラクティブ・グラフィックス用システムHIGPSL（High-level Interactive Graphic Processing System embedded in LISP）の開発を行なっている。ユーザのプログラミングの労力を極力軽減すること、初心者でも使えるようにグラフィックスの諸概念や言語を単純、明快かつスマートなかたちに整理すること、ハイレベルのシステムであってもユーザが細かいコントロールのできる余地も残しておくこと、が我々の方針である。このシステムはLISPのもつ機能を十分に活用して、グラフィック・システムに共通して必要な種々の機能を内蔵しており、ユーザがアプリケーションの対象の画面・構造に関する情報とインタラクションに関する情報をHIGL（High-level Interactive Graphic Language）とLISPでプログラムし、システムのデータ構造に格納させると、システムはそれを解釈して全体がユーザの目的とするアプリケーション向きのシステムとなって働くもので、ひとつのメタ・システムともいえるものである。

本論文では、まずグラフィック・システムとしてのHIGPSLの特徴をのべ、次にその構成と内容、HIGLについて詳しくのべる。

§1．HIGPSLの特徴

§1．1　LISPの機能

HIGPSLはLISPに埋めこまれている。何故LISPのような一般的とはいいかねる言語を用いるのか、という批判があるかもしれない。我々がLISPを用いた理由は、ひとつには、我々のところにはLISPしかハイレベル・システムを実現するのにふさわしい言語プロセッサがなかったためであり，また人工知能やパターン認識、グラフ処理などの分野でのLISPの有用性を知っておりその分野でのグラフィックスの活用を期待するためでもある。しかし一番大きな理由は、次の節にのべるように、LISPのもつ機能がおそらく他の言語プロセッサからは得られないほど優れたもので、GSPと共用すると画面の表示、データ構造、アテンション、ハンドリングを統一した言語で記述できるところにある。他の言語でハイレベルのシステムを作る場合には、そのまま移すことは困難

であろうが、ＨＩＧＰＳＬの考え方は十分示唆を与えるものと思う。

ＬＩＳＰのもっ一般的機能を次にあげる。

① 記号処理、リスト処理用言語である。
② 関数の nesting と再帰的な使用ができる。
③ storage の自動的割付けと、garbage collection が自動的に行なわれる。
④ プログラムとデータの形式が同一。関数をプログラムの実行中に定義し、組立て、実行できる。
⑤ インタープリターであるので、その上にインタープリターを組立てること、およびデバッグが容易。

§1、2 ＨＩＧＰＳＬの特徴

インタラクティブ、グラフィックスのシステムとしてのＨＩＧＰＳＬおよびＨＩＧＬの特徴を、次の7つの観点からのべ、ＬＩＳＰの機能がどのように活用されているかを示す。

i） プログラミング言語

ＧＳＰで coding されたプログラムを眺めてまず感ずることは、ＣＡＬＬ文の連続でパラメータが沢山あって、プログラムが何をしているのか直観的に分らない、ということである。またグラフィックスではよく必要となる移動や回転、拡大縮小などのためには画面表示のためのディスプレイ・ファイルを作り直さなければならないので、プログラミングの手間は大変である。また画面に表示される図形要素につける名前も数値であるという制約があり、その管理はユーザに任されている。

ＨＩＧＬでは、表現の簡潔さに特に留意し、プログラムの documentation を良くした。特にＬＩＳＰ特有のリスト表現によりプログラムを高水準で記述できるので表現の形態がきれいで分りやすい。同じ集合に属する要素にはユーザはひとつひとつ名前をつける必要はなく、その集合の名前でプログラムを書けばよいのでスマートな記述ができる。これはシステムの方でひとつひとつの要素に名前をつけて、ユーザの与えた情報との関連づけをするためである。ＨＩＧＬプロセッサーはユーザにより与えられたデータ構造の内の情報を解釈しながら働く高水準のインタープリタであるが、ＬＩＳＰインタープリタの持つ入出力、データ構造、関数、解釈ルーチン，などをそっくり借用しているわけで、システムの製作とデバッグが容易である。このことからも base language の重要性は明らかであろう。

ii） 作図能力

グラフィックスではデータ構造内の要素が図形のかたちで，その要素の位置，方向，スケーリング、ファクタに関するデータやユーザの指示に従った移動，回転，拡大縮小、投影，などの座標変換の演算、あるいはさらに必要ならば hidden line 除去、切断、などを行って画面上に表示される。これは無限の広がりをもつ2次元または3次元空間にある図形をグラフィック・ディスプレイの画面の小さな窓を通して見るという形式であるから、窓の外にある線分は表示しないという scissoring の操作が必要である。ＨＩＧＬでは各要素に座標系をもたせることができ、上の操作を簡潔に記述できるが、使用しているＧＳＰがディスプレイ・ファイルのデータをプログラムで自由にかえられるようにはなっていないので、画面の変更にはやゝ時間がかゝる。なお表示図形を記述するには procedure 形式が便利なので関数であらわす。

iii) データ構造

どのデータ構造を用いるかはアプリケーションによって選ばれるべきもので、ユーザに特定のものの使用を強制するのは望ましくない。また汎用データ構造は一般にデータ処理に時間がかかる。LISPではLISPのformatにのっとった任意のリストを作れるfree storageと，arrayが使える。HIGPSLではこれら両方を活用しており，システム用のデータ構造とユーザが用いるものとを分離して、データの処理を高速化した。arrayは幾何学的図形を対象とする場合の数値データの記憶に用いている。

iv) インタラクション用の画面形成

画面に表示される図形はデータ構造内の要素と対応のとれている必要があり、画面の管理やライトペンによる割込処理のために入れ子構造に構造化されて管理される。その時識別のために図形要素に重複しない名前（GSPではcorrelation valueという）をつける必要がある。我々の用いているGLISPでは、システムが発生した要素名のアトムのhead cellの番地をその名前としてGSPに渡しているので、図形要素の属性のコントロールや割込み処理をLISPの表現で直接行えて便利である。ユーザが画面の形成や変更を行なうときはHIGLで簡単に記述でき、あとはHIGPSLの方でやってくれる。

v) アテンション・ハンドリング

インタラクションの場面は、人間の動作に対する計算機の反応という、有限状態オートマトンのstate diagram表現に似た捕え方が多くの場合適当である。また人間の動作と計算機の処理の2つの並列プロセスの同期をとるところとも考えられる。PL/IのONCONDITIONのような表現の方が具合の良いときもある。通常計算機の処理速度は人間の動作にくらべ高速なので多重割込のおこる心配のあること、また人間の誤動作を防ぐこと，のため制御可能な図形要素の属性のコントロールを厳密に行なう必要がある。ここで制御可能とはライトペンによる検知性と、可視性のコントロールのできることを意味する．HIGPSLでは、ユーザがHIGLで記述したState diagram表現とシステム用データ構造に登録された関係をもとに、図形要素の属性のコントロールと、割込情報から遷移選択枝の決定を自動的に行っている。しかしユーザが手のこんだ細かいコントロールを行なう余地も残してある。またLISPでは関数の再帰的使用のできることを利用して、アテンション・ハンドラーを再帰的によぶことにより多重state diagramの取扱いができる。state diagramは自由に追加、変更が行なえるようになっている。

vi) マン・マシン・システムとして必要なfacilities

これには次のようなものがある。

① 計算機から必要に応じてstateごとに言語情報を画面に表示すること　システム作りにおいてデバッグのさいに計算機が今どのStateにいるかを表示するのはプログラムのチェックに大いに役立つ。また他人の作ったグラフィック・システムはどう使えばいいのか分らないものなので、Stateに応じてユーザに助言を与える誘導型のシステムへと変えられることも必要である。HIGLではState diagramの記述のさいに、表示する言語情報をも記せるようになっている。

② UNDO　ある動作をしたあとで、その動作がなかったものとしてもとにもどりたい場合には、HIGPSLではそれ専用のファンクション・キーを押すと、前の動作に対する計算機の反応の中の関数で組立てた、逆の働きをする表現

が実行されて、Side effectsが除かれ、もとのStateにもどる。もっとglobalなレベルで行ないたいときは、そのための関数が用意してあるので、それをよぶ。

③ マクロ定義とマクロ・コール　このための関数が用意されている。

④ Flickering　まだとり入れていない。

⑤ ハード・コピー　現在準備中。

インタラクティブ・システムでとくに大事な計算機の応答の早さの点で、HIGPSLはやや問題があるが、コンパイラが完成すると改善されるものと期待している。

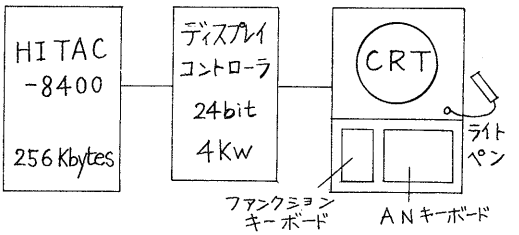
(vii) ファイル

実際のアプリケーションに用いられるためには、ファイルはかならず必要である。またLISPはメモリ領域を沢山使用する。そこでGLISPをファイルの使用とSwappingができるよう拡張作業中である。

§2. 使用しているシステムとHIGPSLの構成

§2.1 ハードウエア

第1図に我々が現在使用しているシステムのハードウエアの構成を示す。この図から分るように、グラフィックス関係の割込み源としては、ライトペン、ファンクション・キー、英数字キー・ボードのEMキーとCLキーがある。ライトペンの機能としてはpicking, pointingおよびtrackingができる。

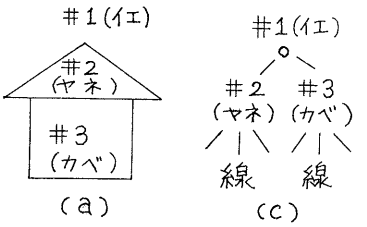


第1図　ハードウエアの構成

§2.2 ソフトウエア

i) GSP

外部使用についてはIBM1130/2250GSPにもとづいている。サブルーチンは49個、全体の大きさは約40Kバイトである。このGSPの使用上の制約として今まであげたことの他に、制御可能エレメントが画面管理の木構造(入れ子構造)のなかで1レベルしか使えないことがある。たとえば第2図(a)のような図形を(b)のようなCALL文により(c)のような木に構造化している。



(b)

```
Begin Element #1
  Begin Element #2
    Move Position
    Plot Line
  End Element #2
  Begin Element #
  End Element #3
End Element #1
```

第2図　GSPの木構造の例

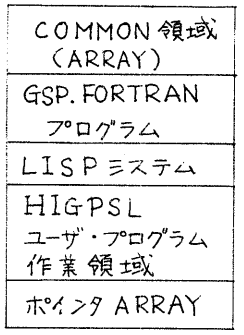
ii) GLISP

我々の用いているGLISPでは、他の言語で書かれたプログラムを呼べることを利用して、GSPの各サブルーチンをLISPの関数として定義してあるので、ユーザはLISP上から容易にグラフィック・ディスプレイを使うことができる。LISPとFORTRANの間のパラメータの受渡しは、FORTRANのCOMMON領域上にLISPの方からarrayを定義できるので、そこを通してデータの授受が行なえる。このarrayはLISP1.5のポインタのarrayとは別の領域にあり、働きも異なる。この領域にはGSPが使用するImage Construction Area (ICA)やGraphic Control Area (GCA)もおかれている。LISPとFORTRANの間で受渡しす

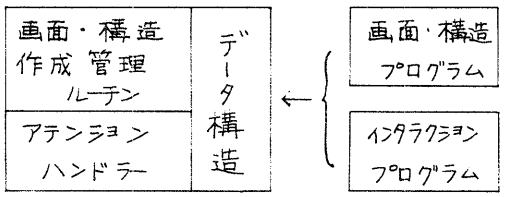
るデータの種類は，実整数 文字，および ＬＩＳＰ の atom をさすポインタである。 第３図にコア記憶領域の割付けを示す。

§２．３ ＨＩＧＰＳＬの構成

第４図はＨＩＧＰＳＬの構成を示したものである。ＨＩＧＰＳＬは大別して，システム用データ構造，画面，構造作成管理ルーチン，アテンション・ハンドラーからなっている。ユーザは図のように画面，構造に関するプログラム，およびインタラクションに関するプログラムを，ＨＩＧＬおよびＬＩＳＰで書いて与えると，ＨＩＧＰＳＬ全体がユーザの目的とするアプリケーション向きのシステムとなって働く。画面・構造に関する記述は，大別して対象のもつ構造と画面上の構造，および位置，方向，スケーリング・ファクタなどの空間的情報と，表示のための図形定義関数からなる。§３・２にのべるように，幾何学的図形対象とする場合と，非幾何学的な位相的論理関係のみを問題とする場合とでは記述の仕方が全く異なる。インタラクションを記述するプログラムは，state diagram の記述と，その中で計算機の反応の内容を記述した関数（Program Block，略してＰＢと，Instruction for EXecution，略してＩＥＸ）群からあらわされる。画面，構造，空間の記述と State diagram の記述には，かなり複雑な syntax がある。表示図形定義関数とＰＢ，ＩＥＸはＨＩＧＬが用意した関数をＬＩＳＰの関数とまぜてプログラムする。



第３図 記憶領域の割付け



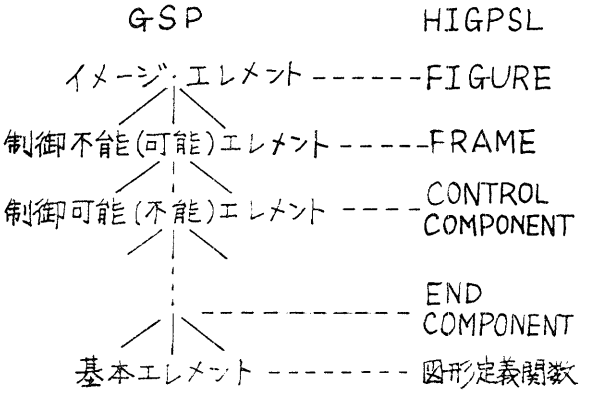
第４図 ＨＩＧＰＳＬの構成

§３．画面・構造の記述と作成管理ルーチンの働き

§３．１ 画面の木構造

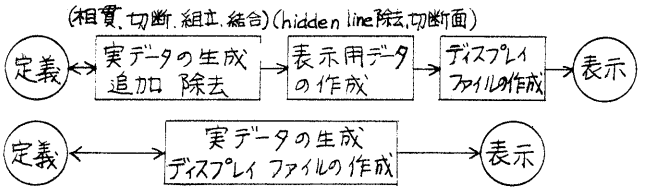
ＨＩＧＰＳＬでは，第５図のようなＧＳＰの木構造の各レベルに同図右のような一般的呼称と関数が対応する。ＦＲＡＭＥとは画面のある領域を通して眺められる世界を指し，ＣＯＭＰＯＮＥＮＴとは制御不能エレメントの一般的呼称で，特に上に"ＣＯＮＴＲＯＬ"とつくと制御可能を意味する。以後，制御可能なＦＲＡＭＥとＣＯＭＰＯＮＥＮＴのことをＣＮＴＦＲとＣＮＴＣＭＰ，ＣＯＭＰＯＮＥＮＴをＣＭＰ，図形定義関数とそれをもつＣＯＭＰＯＮＥＮＴのことをＰＩＣＦＮとＥＮＤＣＭＰ，と略称する。またそのあとに一名とつけて，それのひとつである集合的な名詞を指す。ＨＩＧＬのプログラムはこれを用いて記述される。

ＦＲＡＭＥは画面における窓の座標を与えて scissoring の操作の指定が可能で，窓自身の固有の目盛をつけかえることにより平行移動と拡



第５図 ＨＩＧＰＳＬにおける画面の木構造

大縮小が自由に行なえる。ＨＩＧＬでは画面の各要素に固有座標系をもたせることができるので、木構造の下の要素を上の要素に対して自由に移動、回転、スケーリングさせることができる。



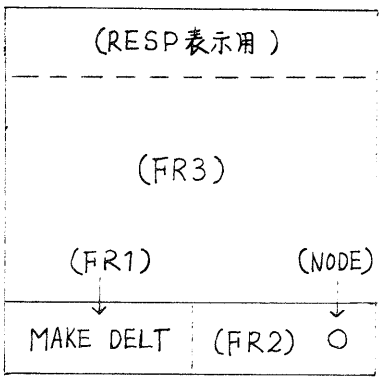
オ６図　定義から表示までの過程

§３．２．ＨＩＧＬによる画面構造の記述とシステムの動作

前にものべたが、幾何学的な図形を対象とする問題と位相的論理関係が意味をもつ問題とでは、画面・構造の記述と処理に関して本質的なちがいがある。オ６図はこのふたつの問題について定義から表示に至る迄の処理のステップを示したものである。これから分るように、後者の問題ではユーザの与える定義からそのまゝ実際の要素とその構造、およびディスプレイ・ファイルが生成される。画面に表示される図形は定義の木の葉の部分に位置し、人間との情報交換のための媒体にすぎない。これに対し前者の問題では、頂点も重要な要素であり、また回転や投影など座標変換も多い。また定義の構造は木ではなく、定義から生成される実際の要素間の構造に、相貫、切断、結合などの操作により変更が加えられたり、さらにはhidden lineの除去、切断面の表示など現実に存在する線分そのままではない表示用のデータを求めてから、実際のディスプレイ・ファイルが作成される場合も多い。

ｉ）　位相的論理関係が意味をもつ問題

例としては簡単すぎるかもしれないが、オ７図のような画面を表示するためのＨＩＧＬによる定義の記述は下のようになる。ＦＲ１は制御可能なＦＲＡＭＥで、２つの文字メニューをＣＯＭＰＯＮＥＮＴとしてもつ。ＴＥＸＴはその第１引数の単語を表示する関数である。ＦＲＡＭＥ　ＦＲ２は制御可能な図形メニューＮＯＤＥをもち、その図形定義関数は一番下に定義されている。ＦＲＡＭＥ　ＦＲ３はＮＯＤＥが作られていくところで、画面の座標系（中心が原点で、ＸＹとも５１１～－５１２）からみた窓枠の座標が与えられscissoringが行なわれる。また窓



オ７図　簡単な画面の例

```
DECOM((FIG1
(CNTFR (FR1 (AT -400 -400)
        (CMP (MAKE (AT 0 0)(PICFN (TEXT MAKE 1)))
             (DELT (AT 200 0)(PICFN (TEXT DELT 1))) )))
(FRAME (FR2 (AT 400 -400)(CNTCMP (NODE (PICFN (NODE)))))
        (FR3 (SCISSORING -512 -312 511 361) (WIDOW 0 0 100 60)) )
(PICFN (PROG ()(MOVEA (511 -312))(LINEA (-512 -312))
        (LINEI (0 -200))(LINEA (512 -512)(511 511)(-512 511)(-512 -312)) ))
))
DEFINE ((NODE (LAMBDA ()(PROG ()(MOVEA (0 10))
        (LINEA (7 7)(10 0)(7 -7)(0 -10)(-7 -7)(-10 0)(-7 7)(0 10)) )))))
```

の目盛がＷＩＮＤＯＷで与えられる。ＤＥＣＯＭの最後のものは画面のわくを描くための図形定義関数である。ＭＯＶＥＡ，ＭＯＶＥＩ，ＬＩＮＥＡ，ＬＩＮＥＩの語尾のＡとＩはＥＮＤＣＭＰの座標系における絶対モードと増分モードをそれぞれ示す。ＭＯＶＥはblankingで、ＬＩＮＥは眼に見える線をひく。ＤＥＣＯＭは定義を利用しやすい形に分解してシステム用データ構造にしまう。

ＭＫＦＩＧ（ＦＩＧ１）という命令が与えられると、ＨＩＧＰＳＬはこのデータ構造内の定義をたどって、実際の要素名を関数ＧＥＮＳＹＭによって発生し、下にのべるような関係をＦＲＡＭＥ要素名のもとに登録し、同時に座標変換の計算を行なって、第２図（ｂ）のような入れ子構造に展開された表示命令のあつまりを生成する。ＦＲＡＭＥの要素に登録される関係には次のようなものがある。

① 定義および一般のＣＯＭＰＯＮＥＮＴ名と実際の要素との対応

② 要素どうしの上下関係　③ ＦＲＡＭＥの要素と制御可能な要素との関係

これらはglobalなＵＮＤＯに都合の良い形式に納められている。ユーザが検索できるようにしておく必要がある。ＧＥＮＳＹＭはカウンターのようなもので、よばれるたびに新しい要素名（Ｇｏｏ１，Ｇｏｏ２，--）を発生する。ユーザがある要素に別のＦＲＡＭＥ名やＣＯＭＰＯＮＥＮＴ名をつけることも可能である。

この他、インタラクションの時点において、画面およびデータ構造に、新要素追加、既存要素の除去のための関数ＥＸＴＥＮＤとＤＥＬＥＴＥ，およびＵＰＤＡＴＥが関数として用意されている。

画面・構造記述プログラムのsyntaxの一部を下に示す。ここで［］は省略してもよいもの，｛|｝はその中の少くとも一方が必要なものをあらわす記号である。（制御可能については厳密でない）

```
<FIG DESC>    ::=(<FIGURE> <FRAMEPL>[<PICFN LIST>]
<FRAMEPL>     ::={(CNTFR <FRAME LIST>)|(FRAME <FRAME LIST>)}
<FRAME LIST>  ::=<FRAME LIST><FRAME DSCR>|<FRAME DSCR>
<FRAME DSCR>  ::=(<FRAME><FRAME SPEC><COMP SPEC><CMPPL>
<CMPPL>       ::={(CNTCMP <COMP LIST>)|(CMP <COMP LIST>)}<PICFN>
<COMP LIST>   ::=<COMP LIST><COMP>|<COMP>
<COMP>        ::=(<CMP><CMP SPEC><CMPPL>)
<FRAME SPEC>  ::=[(DIMENSION <NUM>)][(SCISSORING <CO-LIST>)]
                 [(WIMDOW <CO-LIST>)][(DISTANCE <NUM>)]
<COMP SPEC>   ::=[(AT <NUMLIST>)][<ROTATION>]|
                 [(SCALE <NUM>)][(PARAM <PARAMLIST>)]
<PICFN>       ::=(PICFN <S-EXPR>)
```

ii）　図形自体を対象とする問題

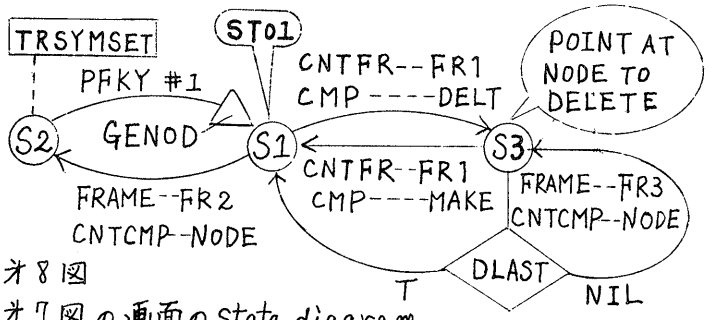
六面体のような立体をそのまゝ表示する例を次頁に記す。ＤＥＧＦは図形の定義を与える関数で、ＤＣＬは属性の宣言、ＲＥＬは領域、面、稜、頂点の各レベルの項目の関係を両方向に記述したものである。次のＥＸＴＥＮＤＧにより、上の定義から現実の図形要素と構造が生成され、ＢＯＸの要素をＦＲＡＭＥ　ＦＲ４からみた斉次行列がデータとして与えられ、Ｐ１～Ｐ８に各頂点の変換前と後の座標値をおさめたＡＲＲＡＹの列番号が付与される。次のＤＥＣＮＴは画面管理のためで、ＢＯＸは制御可能なことを示す。最後のＭＫＦＧによりＦＲＡＭＥ　ＦＲ４に存在する立体の表示ファイルが作成される。hidden line除去のときはＤＣＬを除いて、可視線分の上下関係と、ＬＩＮＥとの対応づけを与えてからよぶ。

```
DEGF((BOX
(DCL (LINE L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12))
(REL ((FACE SOLID)((BOX F1 F2 F3 F4 F5 F6)))
     ((LIN FC)((F1 L1 L2 L3 L4 L1)(F2 L1 L5 L6 L7 L1)
                ----------(F6 L7 L8 L9 L2 L7)))
     ((PNT LN)((L1 P1 P2)-------(L12 P7 P8))) ) ))
EXTENDG(FR4 (BOX (AT 100.0 0.0 0.0)(ROTX 45.0)
          (COORD (P1 10.0 20.0 30.0)---(P8 -10.0 -20.0 -30.0)) ))
DECNT((BOX (CMP (LINE (PICFN (LINEDRAW))))))
MKGF (FR4)
```

§4. インタラクションの記述とアテンション・ハンドラーの働き

§4.1 state diagram のHIGLによる記述

HIGPSLのユーザは，自分が対象としている問題のインタラクションの場面のstate diagramを紙の上に描いてみてから，それをHIGLで記述してHIGPSLに与える。HIGPSLはそれを分解してシステム用データ構造に格納する。オ7図の画面のインタラクションのstate diagram（オ8図）をHIGLで記述すると下のようになる。図形メニューをライトペンでpickし，その上に表示されるtrancking symbolをFR3の適当な位置に移動してファンクション・キー1を押すと，NODEがそこに表示される。メニューDELTをpickしてから，次々のFR3上のNODEをpickすると次々と消えていく。全部無くなればS1にもどる。途中S1にもどりたいときはメニューMAKEをpickする。STATEはstate diagramの記述をデータ構造にしまうHIGLの関数で，下側5



オ8図

オ7図の画面のstate diagram

```
STATE(
(S1 (RESP ST01)
    (PENHIT (FR1 (FR1 (DELT (NEXT S3)))) (FR2 (NODE (NIL (NEXT S2))))))
(S2 (PB (TRSYMSET 400 -400 FR2))
    (PFKY (1 (IEX (GENOD))))  )
(S3 (RESP ¥¥*POINT AT NODE TO DELETE*)
    (PENHIT (FR1 (FR1 (MAKE (NEXT S1))))
            (FR3 (NODE (NIL (TEST (DLAST NODE S1 S3))))) ) ) )
DEFINE(((GENOD (LAMBDA ( ) (PROG (A)
    (SETQ A (TRFIN FR3)) (EXTEND FR3 (NODE (AT (CAR A)(CADR A)))
)))))
DEFLIST(((DLAST (LAMBDA (X AL) (PROG ( ) (DELETE GFR GCNT)
    (COND ((NULL (RASSOC GFR (QUOTE ATRB) (CAR X))) (RETURN (CADR X)))
          (T     (RETURN (CADDR X))) ) )))) FEXPR)
```

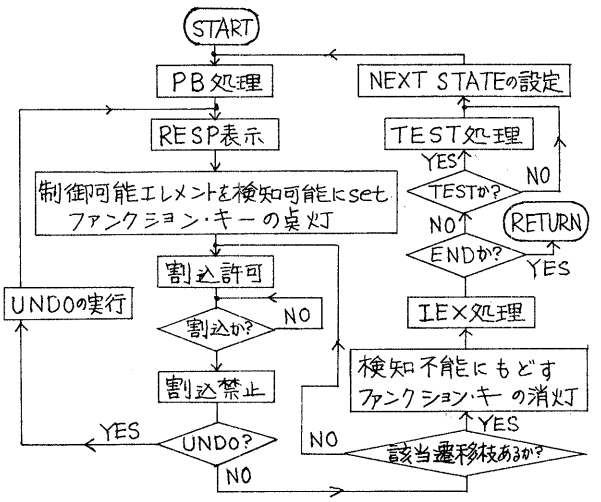
行はＮＯＤＥの追加と消去を行うＧＥＮＯＤ（ＩＥＸ）とＤＬＡＳＴの関数の定義である。state diagram においてＲＥＳＰはその state において計算機側から表示される言語情報を示し、ＮＥＸＴはその遷移枝が選ばれた場合の次の state，ＴＥＳＴはその結果により次の state が異なる判断の関数を示す。ＰＥＮＨＩＴとＰＦＫＹはそれぞれライトペンとファンクション・キーの許容割込を示し，ＰＥＮＨＩＴのときはＦＲＡＭＥ，ＣＮＴＣＭＰ，ＥＮＤＣＭＰの順に名前を記述し、ファンクション・キーのときはキー番号を指定する。ＴＲＳＹＭＳＥＴとＴＲＦＩＮはtracking symbol の表示と，消去および座標値の検出，ＥＸＴＥＮＤとＤＥＬＥＴＥは画面とデータ構造への要素の追加と除去，ＲＡＳＳＯＣはデータ構造の検索，をそれぞれ行うＨＩＧＬの関数である。またＧＦＲとＧＣＮＴはアテンション・ハンドラーがライトペン割込のときに見出したＦＲＡＭＥとＣＮＴＣＭＰの具体的な要素名である。

<ATTENTION> ::= [<LIGHTPEN> | <PFKY>] [<EMKY>]
<LIGHTPEN> ::= (PENHIT <FRAME LIST>)
<FRAME LIST> ::= <FRAME LIST> <FRAME DESC> | <FRAME DESC>
<FRAME DESC> ::= (<FRAME> <CNTCMP LIST>)
<CNTCMP LIST> ::= <CNTCMP LIST> <CNTCMP DESC> | <CNTCMP DESC>
<CNTCMP DESC> ::= (<CNTCMP> <ENDCOMP LIST>)
<ENDCMP LIST> ::= <ENDCMP LIST> <ENDCMP DESC> | <ENDCMP DESC>
<ENDCMP DESC> ::= (<ENDCMP> <REACTION>) | (NIL <REACTION>)
<REACTION> ::= [(IEX <S-EXPR>)] (NEXT <STATE>) | END |
(TEST <S-EXPR>)

SYNTAXの一部

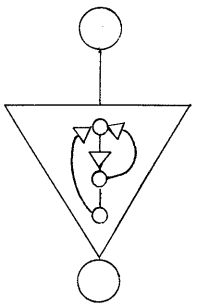
§4，2 アテンション・ハンドラー

アテンション・ハンドラーはシステム用データ構造に格納されているユーザから与えられたインタラクションの記述を解釈しながら動く。そのときシステム用データ構造にしまわれている画面構造に関する関係を full に活用する。第9図はアテンション・ハンドラーのプログラムのフローチャートであり、ある state から次の state への遷移はこのプログラムのループのひとめぐりに対応する。この関数は state diagram の入力 state を引数として呼ばれる。ある state においてＰＢとＲＥＳＰの指定があればそれを行い、ライトペン割込の指定があればそのＦＲＡＭＥとＣＮＴＣＭＰの名前からシステム用データ構造を検索して相当する制御可能エレメントを見出し，その属性を検知可能に設定する。ファンクション・キー割込のときは指定キーのランプを点灯する。それから割込を許可し，割込のおこる迄待つ。１ステップ前にもどるＵＮＤＯ用のファンクション・キーは常時割込めるようにアテンション・ハンドラーが管理していて，もしその割込がおこったときには，前のＩＥＸまたはＰＢの関数の結果生じた side effectsを打ち消す関数が実行され，前の state にもどる。ライトペ



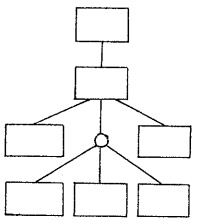
第9図　アテンション・ハンドラーのフローチャート

ンのときは実際に割込んだＣＮＴＣＭＰとＥＮＤＣＭＰの要素名とシステム用データ構造の中の関係から，ＦＲＡＭＥ名やＣＮＴＣＭＰ名，ＥＮＤＣＭＰ名が分る。ファンクション・キーのときはキー番号が分る。state diagram の該当遷移枝が決定されると，検知可能にした要素を不能にもどし，ファンクション・キーのランプを消してから，該当遷移枝にあるＩＥＸやＴＥＳＴの処理を行ない，次のstate に移る。ＥＮＤならそのアテンション・ハンドラーから戻ることにより今のstate diagram から抜け出す。第１０図に示すように，ＩＥＸまたはＰＢの中でアテンション・ハンドラーを再帰的に呼ぶことによって，ステート・ダイアグラムの多重使用が可能となる。このような使用法としては次の２通りが考えられる。



第10図　多重 State diagram

ｉ）　ユーザが１つのアプリケーション・システムをこしらえる場合，画面を幾つも用意して，第１１図のように画面群に木構造を導入して管理するのが普通である。ある画面からその下のノードにあたる画面をよぶことは，そのステート・ダイアグラムの途中でアテンション・ハンドラーを再帰的によんで，下の画面のステート・ダイアグラムにコントロールを移すことによってなされる。



第11図
画面群の木構造

ⅱ）　よく用いる一連のaction-reaction のステート・ダイアグラムをシステムで用意し，それにもとづいて動くアテンション・ハンドラーを組込み関数として解放してユーザの便宜をはかる。もちろんユーザ自身が自分の目的のためにこのような関数を定義して使うこともできる。

むすび　現在ＨＩＧＰＳＬの完成度は７０％位であろう。近い将来ファイル，コンパイラなどＧＬＩＳＰの機能が拡張されて，ユーザはＨＩＧＰＳＬにより簡単にディスプレイ装置をつかいこなして自身の研究の能率をあげる日が来るものと信じる。

おわりに日頃御指導，御援助頂く，石井ソフトウエア部長，加藤情報システム研究室長，同研究室大石君に感謝致します。また仕事のお手伝い頂いた慶応大学岡田君，成蹊大学山腰君，清書して頂いた升森さん，蜂須賀さんに感謝します。

参考文献

1．電子技術総合研究所彙報，コンピュータ・グラフィックス特集号　３７巻１，２号（１９７３）

2．Newman, w. A., Display Procedures Comm. ACM 14 10 (Oct. 1971), 651-660

3．————; A system for interactive graphical programming, AFIPS. 1968, SJCC., pp-308-315