グラフィクスとＣＡＤ 12－4
（１９８４．３．１６）

ＧＫＳによるグラフィクス・プログラミング

山口 泰，木村文彦 （ 東大・工 ）

１． はじめに

ＩＳＯグラフィクス標準はＧＫＳ．version 7.2 (1982, 12) （注[1] ）をもってＤＩＳとなり、世界的な標準として普及する日も近いと思われる。その基本的な思想と概要は各種の文献などにより紹介され、広く知られてきている。然るに、ＧＫＳの実際的な利用となると我が国では、未だ研究レベルにおいても初期的な段階にあるように思われる。本稿ではこの利用法への端緒としてＧＫＳによるプログラミングについて考えてみる。更にＧＫＳプログラム・パッケージを用いて、実際にＧＫＳによるグラフィクス・プログラムを書くとともにその実行例を示す。これを踏まえた上で、ＧＫＳの特徴であるワークステーションの機能に重きをおいて、ＧＫＳの利用法と今後の姿について考察を行なう。

２． ＧＫＳを用いたプログラミング

２－１． ＧＫＳ・ステート

ＧＫＳにおいてはそのステートが非常に重要位置を占めており、すべてのオペレーションはステートと密接な関係を持っている。ＧＫＳ・ステートは５つの状態より成り、各ステートとステート間の移行は図２－１に示すようになっている。図で示された特定のオペレーションだけがステート間の移行に関与しており、他のオペレーションはステートの変更を行なわない。以降では、各ステートを次に示すように略す。

ＧＫＣＬ ＝ ＧＫＳ closed
ＧＫＯＰ ＝ ＧＫＳ open
ＷＳＯＰ ＝ At least one workstation open
ＷＳＡＣ ＝ At least one workstation active
ＳＧＯＰ ＝ Segment open

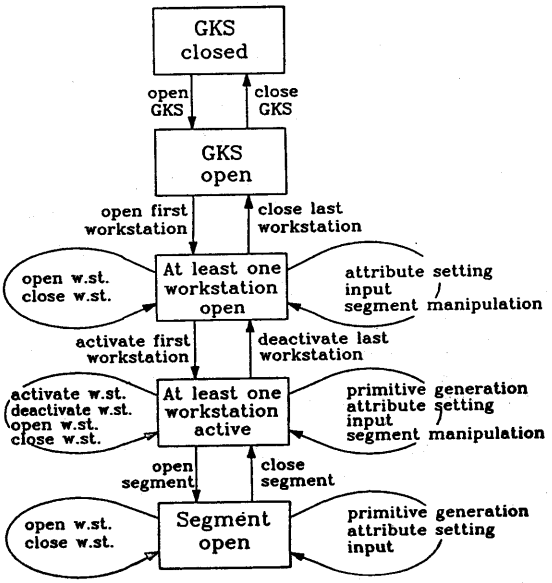

図２－１ ＧＫＳ・ステート

注[2]

２－２． 座標系とその変換

ＧＫＳではＷＣ，ＮＤＣ，ＤＣという３つの座標系とその間の変換によって位置情報を操作する。各座標系とその関係を図２－２に掲げる。

normalization transformationと workstation transformation はそれぞれＷＣ上のwindowとＮＤＣ上のviewport ＮＤＣ上の workstation window とＤＣ上の workstation viewportという長方形の領域を指定し、その間の写像として定義されている。出力命令や入力命令に応じてこの写像を実行する。segment transformationは２ｘ３の変換行列によって表現されている。セグメント操作命令に応じてこの行列をセグメントに作用し、これを行なう。これらの変換の具体的な指定法は出力命令、入力命令、セグメント操作命令と深く関連しているので、それぞれの説明の際に順次触れていく。

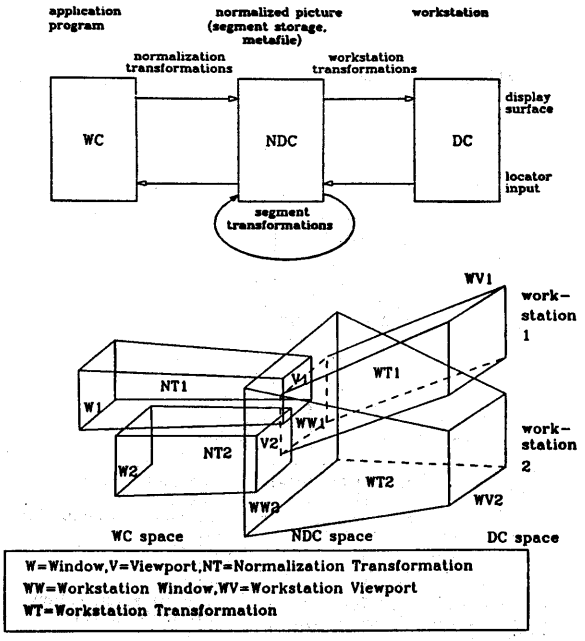

図２－２　各座標系とその関係

注[2]

２－３．　出力命令

ＧＫＳに於いては、出力命令は出力基本命令とその出力の属性指定命令の２つの命令から成っている。

出力基本命令として以下の６つのOutput Primitiveが用意されている。

1)　ＰＯＬＹＬＩＮＥ

2)　ＰＯＬＹＭＡＲＫＥＲ

3)　ＴＥＸＴ

4)　ＦＩＬＬ　ＡＲＥＡ

5)　ＣＥＬＬ　ＡＲＲＡＹ

6)　ＧＥＮＥＲＡＬＩＺＥＤ　ＤＲＡＷＩＮＧ　ＰＲＩＭＩＴＩＶＥ

属性指定命令としては bundle を用いた指定法と individual な指定法の２種類の指定法がある。 bundle を用いた指定法はワークステーションの上での属性指定法であり、これを workstation dependent な属性指定法であるという。これに対して individual な指定法はワークステーションとは無関係の指定法であり、workstation independent な属性指定法であるという。このため、bundle 指定の可能なＧＫＳ・ステートはＷＳＯＰ，ＷＳＡＣ，ＳＧＯＰであるが、individual な指定の可能なＧＫＳ・ステートはＧＫＯＰ，ＷＳＯＰ，ＷＳＡＣ，ＳＧＯＰとなっている。指定法の選択は、Ａspect Ｓourse Ｆlag （ＡＳＦ）の切り換えによって行なう。ＡＳＦの初期設定はインプリメンタの判断に任されている。（本稿の例ではbundle指定とする。）

ＰＯＬＹＬＩＮＥの場合には、出力命令のプログラムは以下のような形になる。ここでは、命令コードと命令内容の対応がつきやすいPascal-binding でのプログラムを書く。紙面の制約もあるので、主たる操作であるＧＫＳの命令コードのみを引数を省略した形で記す。

bundleを用いて属性を指定する場合のプログラムを示す。

```
L000    PROGRAM EXAMPLE1;
L001    begin
L002      OPEN_GKS;
L003      SET_WINDOW;
L004      SET_VIEWPORT;
L005      SELECT_NORMALIZATION
                _TRANSFORMATION;
L006      SET_POLYLINE_INDEX;
L007      OPEN_WORKSTATION;
L008      SET_WORKSTATION_WINDOW;
L009      SET_WORKSTATION_VIEWPORT;
L010      SET_COLOUR_REPRESENTATION;
L011      SET_POLYLINE_REPRESENTATION;
L012      ACTIVATE_WORKSTATION;
L013      POLYLINE;
L014      DEACTIVATE_WORKSTATION;
L015      CLOSE_WORKSTATION;
L016      CLOSE_GKS
L017    end.
```

プログラムの各行では以下の操作を行なっている。

L002でＧＫＳを起動する。

L003～L005でnomalization transformation を設定する。

L006でPOLYLINEの属性指定に利用するbundleを指定する。

L007でワークステーションを作動可能する。

L008,L009 でworkstation transformationを設定する。

L010,L011 でPOLYLINEの属性をbundle内に書き込む。

L012でワークステーションを表示可能にする。

L013で実際にPOLYLINEを描く。

L014～L016でワークステーション及びＧＫＳを停止する。

上の例では、属性指定をすべて bundle を用いて行なっている。そこで individual な指定法を用いてこれと同じプログラムを書く場合には、次のようになる。

```
L000    PROGRAM EXAMPLE2;
L001    begin
L002      OPEN_GKS;
L003      SET_WINDOW;
L004      SET_VIEWPORT;
L005      SELECT_NORMALIZATION
                _TRANSFORMATION;
L006      SET_ASPECT_SOURSE_FLAG;
L007      OPEN_WORKSTATION;
L008      SET_WORKSTATION_WINDOW;
L009      SET_WORKSTATION_VIEWPORT;
L010      SET_LINETYPE;
L011      SET_LINEWIDTH_SCALE_FACTOR;
L012      SET_COLOUR_REPRESENTATION;
L013      SET_POLYLINE_COLOUR_INDEX;
L014      ACTIVATE_WORKSTATION;
L015      POLYLINE;
L016      DEACTIVATE_WORKSTATION;
L017      CLOSE_WORKSTATION;
L018      CLOSE_GKS
L019    end.
```

前の例と異なる部分は、L006とL010～L013である。

L006では、bundleを指定せずにＡＳＦをきりかえる。

L010～L013では、bundleの内容を書き込むかわりに属性をグローバルな形で指定する。

２－４。　入力命令

ハイレベルなインタラクション操作を効果的に実現するために、入力機能は非常に重要である。ＧＫＳではワークステーションの一部として論理入力デバイスを用意し、これに対応している。論理入力デバイスにはその入力の値に応じて下の６つの input class がある。

1）　ＬＯＣＡＴＯＲ

2）　ＳＴＲＯＫＥ

3）　ＶＡＬＵＡＴＯＲ

4）　ＣＨＯＩＣＥ

5）　ＰＩＣＫ

6）　ＳＴＲＩＮＧ

これらの論理入力デバイスは次の３種のオペレーティング・モードのいずれかのモードで操作される。

1）　ＲＥＱＵＥＳＴ

2）　ＳＡＭＰＬＥ

3）　ＥＶＥＮＴ

入力命令はこの input classの詳細設定、オペレーティング・モードの変更と入力の取り込みの３つの命令より成る。入力命令はワークステーション個々のものであるので、それの許されるＧＫＳ・ステートは殆どの場合、ＷＳＯＰ，ＷＳＡＣ，ＳＧＯＰのいずれかのステートになっている。

input class 詳細の設定は、各input class の初期化という形式で行なう。すべての class に共通したものは、

1）　入力ワークステーションの指定

2）　入力デバイスの指定

3）　入力値の初期値

4）　プロンプトとエコーの型

5）　エコーを行なうにあたっての領域

である。入力値が位置情報のＬＯＣＡＴＯＲとＳＴＲＯＫＥの初期値については、変換に用いる normalization transformation の指定も必要となっている。入力値の幅やその内容といった更に細かな設定には data record という部分に格納することによって、これを行なう。

オペレーティング・モードの変更は各input class についてそれぞれ行なう。ＧＫＳでは、ＲＥＱＵＥＳＴ・モードが初期設定されている。

入力の取り込みは、指定されたオペレーティング・モードに応じて適当な入力コードをもってなされる。

以下では、ＬＯＣＡＴＯＲをＲＥＱＵＥＳＴ・モードで用いるような場合のプログラムを示す。このプログラムでは、ディスプレイ面上の１点を示し、ＷＣ上の座標値として取り込んだ上で、その位置にＰＯＬＹＭＡＲＫＥＲを表示する。

```
L000    PROGRAM EXAMPLE3;
L001    begin
L002      OPEN_GKS;
L003      SET_WINDOW;
L004      SET_VIEWPORT;
L005      SELECT_NORMALIZATION
                _TRANSFORMATION;
L006      SET_VIEWPORT_INPUT_PRIORITY;
L007      PACK_DATA_RECORD;
L008      SET_POLYMARKER_INDEX;
L009      OPEN_WORKSTATION;
L010      SET_WORKSTATION_WINDOW;
L011      SET_WORKSTATION_VIEWPORT;
L012      INITIALISE_LOCATOR;
L013      REQUEST_LOCATOR;
L014      SET_COLOUR_REPRESENTATION;
L015      SET_POLYMARKER_REPRESENTATION;
L016      ACTIVATE_WORKSTATION;
L017      POLYMARKER;
L018      DEACTIVATE_WORKSTATION;
L019      CLOSE_WORKSTATION;
L020      CLOSE_GKS
L021    end.
```

プログラムの各行の意味を説明する。出力命令の例との共通点については省略する。

L006では、normalization transformationの逆変換を行なうためのviewportの優先順位の指定を行なう。

L007では、LOCATOR 入力を行なうにあたっての細かな設定値を定める。

L012でLOCATOR 入力の設定を行なう。ここで逆変換に用いるnormalization transformationをきめる。

L013でLOCATOR 入力の要求とその取り込みを行なう。ここで得られた値をL017において用いる。

L008,L015,L017は、出力命令の前者の例に準ずる。

workstation transformationの逆変換には、workstation に既定されている（唯一の）workstation transformationを用いるためそれを指示してはない。

ＬＯＣＡＴＯＲを各種のオペレーティング・モードで用いる場合には、プログラムでは次のようにモード変更とモードに応じた入力コードの使用が必要となる。

```
L000    {A PART OF PROGRAM EXAMPLE4}
L001      REQUEST_LOCATOR;
L002      SET_LOCATOR_MODE; {to SMAPLE}
L003      SAMPLE_LOCATOR;
L004      SET_LOCATOR_MODE; {to EVENT}
L005      AWAIT_EVENT;
L006      if(input_class=LOCATOR)then
            GET_LOCATOR;
```

２－５。　セグメント操作命令

図形の操作を効果的に行なうためには、図形の部分を１つのまとまりとして扱える必要がある。ＧＫＳではこのまとまりをセグメントという形でもち、これに対して種々の操作を備えている。

セグメントの保持と操作には、２つ型がある。１つはセグメントを個々のワークステーションに蓄え、操作をそれぞれのワークステーションの内において行なうものである。もう１つは、セグメントをワークステーションとは独立に蓄えられ、特別な操作が可能なものである。後者の型のセグメントの保持は、Workstation Independent Segment Storage（ＷＩＳＳ）においてなされる。これに対して前者の型の保持を担う部分を、Workstation Dependent Segment Storage（ＷＤＳＳ）と呼ぶ。

ＷＩＳＳ上のセグメントは、copy,associate,insert といった操作によってviewing pipeline上の適当な位置に取り込むことができる。これらの操作はＷＩＳＳ上のセグメントに対してのみ実行できる。

ＷＤＳＳ、ＷＩＳＳに共通のセグメントに対する操作は、セグメントの生成・消去に関するものとその属性変更に関するものの２つに大別できる。後者では、表示に際してのvisibility,highlighting、pick入力のためのdetectability 、双方に影響するsegment priorityの操作が可能であり、位置や形状の変換などがsegment transformationによって扱えるようになっている。

ここでは、複数のセグメントを生成した上で、その中のいくつかのセグメントにsegment transformationを施して表示するプログラムの例を示す。

```
L000    {A PART OF PROGRAM EXAMPLE5}
L001      ACTIVATE_WORKSTATION;
L002      for i:=1 to n do
L003      begin
L004        CREATE_SEGMENT;
L005        Output_primitives;
L006        CLOSE_SEGMENT;
L007        SET_VISIBILITY {to INVISIBLE}
L008      end;
L009      while notEND do
L010      begin
L011        EVALUATE_TRANSFORMATION_MATRIX;
L012        SET_SEGMENT_TRANSFORMATION;
L013        SET_VISIBILITY; {to VISIBLE}
L014        REDRAW_ALL_SEGMENTS
                    _ON_WORKSTATION
L015      end;
```

プログラムについて説明すると、

L001で、ワークステーションを表示可能な状態にする。これをセグメントの観点から見れば、セグメントを蓄えるワークステーションを定めたともいえる。

L002～L008で、ｎ個のセグメントを生成し、それぞれのvisibilityをinvisible にする。

L011,L012 では、セグメントにsegment transformationを施す。

L013で適当なセグメントのvisibilityをvisible に変え、表示され得る状態にする。

L014で visibleなセグメントを表示する。

3． ＧＫＳプログラム・パッケージ

今回用いたＧＫＳプログラム・パッケージは、アムステルダムのMathmatical Centerで開発され、C-bindingのＧＫＳとしてUNIX 4.2bsd の上にインプリメントされたものである。これを東大大型計算機センターのVAX/UNIXにインストールして用いた。

このプログラム・パッケージでは既存の表示デバイス、あるいは入力デバイスを用いるためのデバイス・ドライバを持っている。これによって、ＧＫＳにおいて本来はワークステーションの機能とされるものをプログラム・パッケージ側でシミュレートし、ＧＫＳとしての機能を実現している。現在、このプログラム・パッケージでは、AED512端末と TEKTRONIX4014端末をサポートするデバイス・ドライバを持っている。本稿のC-bindingによるプログラムの例では、カラー・テーブル及びカラー属性の設定を行なっていない。これは TEKTRONIX4014端末を用いた例であるためで、カラー関係の設定法については、他の設定法より類推されたい。

C言語においては多彩なデータ型が定義可能であり、これによってＧＫＳ上の各変数やＧＫＳステートリスト、ワークステーションリスト上の値を非常に分りやすい形で表現している。（しかし、一方ではこの型宣言があまりに多いために、初めての者にとってはそれが煩雑に感じられかねない面もある。）これらの型宣言については、後のプログラム例でこれが問題となるので、付録にその一部を掲げる。

4． C-bindingのプログラムとその実行例

3節で説明したプログラム・パッケージを用いて、実際にＧＫＳによるプログラムを書く。このプログラムでは、種々のＩＮＰＵＴ機能によりストリングの取り込みやＴＥＸＴ出力の属性の指定をインタラクティブに行ない、それを出力する。text font とtext precisionの指定にあたっては、説明的な観点からＡＳＦの切り換えをしてindivisualな指定法を用いている。（初期設定はbundle指定となっている。）

```
#include "cgksincl.h"

/*GKS_STATE_LIST*/
Gks     *gkss;

/*WORKSTATION_STATE_LIST*/
Wss     *tek;

/*WINDOW*/    /*{xmin,ymin,xmax,ymax}*/
Wrect   wrct3 = { 0.0, 0.0, 1.0, 0.5};

/*VIEWPORT & WORKSTATION WINDOW*/
            /*{xmin,ymin,xmax,ymax}*/
Nrect   nrct3 = { 0.0, 0.0, 1.0, 0.5};

/*WORKSTATION_VIEWPORT*/
            /*{xmin,ymin,xmax,ymax}*/
Drect   drct1 = {  60,  60, 210,1440},
        drct2 = { 240,1470,3000,1620},
        drct3 = { 240,  60,3000,1440},
        drct4 = {1060,2650,2000,2880},
        drct5 = {1060,2420,2000,2650},
        drct6 = {2060,1680,3000,2880},
        drct7 = {1060,1680,2000,2360},
        drct8 = {  60,2580,1000,2880},
        drct9 = {  60,1680,1000,2520};

/*DATA_RECORD_FOR_INPUT*/
Drecord drco1, drco2, drco3, drco4, drco5,
        drco6, drco7, drco8, drco9;

/*WORLD_COORDINATE_AND_VECTOR*/
Wc      pr[4], *pb, v[1];

/*FILL_AREA_REPRESENTATION*/
AreaRep fa;

/*TEXT_REPRESENTATION*/
TextRep tx[1];

/*TEXT_FONT_AND_PRECISION*/
FoPr    fopr;

        /*TEXT_FONT_TYPE*/
String  fonts[10] = {"clarendon.18",
                     "bodoni.i.10",
                     "times.b.10",
                     "meteor.r.12",
                     "meteor.i.10",
                     "stare.i.16",
                     "stare.b.16",
                     "playbill.10",
                     "delegate.b.12",
                     "fix.14"        },
        /*TEXT_PRECISION*/
        prec[3]   = {"STR",
                     "CHAR",
                     "STROKEP"       },
        choic[3]  = {"stop",
                     "again",
                     "more"          };

char    str[32];

Bool    redo=TRUE, more=TRUE;

Locate  loc;

Value   ef, ht, vx = 0.0, vy = 1.0;

Choice  cho1 = 1, cho2 = 1, cho3 = 1;
```

```
show_wrect(wr)
  Wrect *wr;
{
  pb = pr;
  pb->w_x = wr->w_ll.w_x;
  pb->w_y = wr->w_ll.w_y; pb++;
  pb->w_x = wr->w_ll.w_x;
  pb->w_y = wr->w_ur.w_y; pb++;
  pb->w_x = wr->w_ur.w_x;
  pb->w_y = wr->w_ur.w_y; pb++;
  pb->w_x = wr->w_ur.w_x;
  pb->w_y = wr->w_ll.w_y;
}

main()
{
  /*SETTING_OF_DATA_RECORD*/
  drco1.val_rec.val_max = 1.0;
  drco1.val_rec.val_min = 0.0;
  drco2.val_rec.val_max = 1.0;
  drco2.val_rec.val_min = 0.0;
  drco4.val_rec.val_max = 10.0;
  drco4.val_rec.val_min = -10.0;
  drco5.val_rec.val_max = 10.0;
  drco5.val_rec.val_min = -10.0;
  drco6.cho_rec.cho_nr = 10;
  drco6.cho_rec.cho_str = fonts;
  drco7.cho_rec.cho_nr = 3;
  drco7.cho_rec.cho_str = prec;
  drco8.str_rec.str_size = 32;
  drco8.str_rec.str_posn = 1;
  drco9.cho_rec.cho_nr = 3;
  drco9.cho_rec.cho_str = choic;

  /*OPEN_GKS*/
  gkss = open_gks(NULL, NULL, NULL);

  /*NORMALIZATION_TRANSFORMATION*/
  s_window  (gkss->gk_ntran[3], &wrct3);
  s_viewport(gkss->gk_ntran[3], &nrct3);
  sel_cntran(gkss->gk_ntran[3]);
  s_vip(gkss->gk_ntran[3], gkss->gk_ntran[0], TRUE);

  /*SET_FILL_AREA_INDEX*/
  s_fa_i(2);

  /*SET_ASF_FOR_TEXT_FONT_AND_PRECISION*/
  s_asflag(0100);

  /*SET_TEXT_INDEX*/
  s_tx_i(3);

  /*OPEN_WORKSTATION*/
  tek = open_ws(NULL, NULL, "tek");

  /*WORKSTATION_TRANSFORMATION*/
  s_w_wind(tek, &nrct3);
  s_w_view(tek, &drct3);

  /*SET_FILL_AREA_REPRESENTATION*/
  fa.fa_is = HOLLOW;
  fa_rep(tek, 2, &fa);
  show_wrect(&wrct3);

  /*ACTIVATE_WORKSTATION*/
  activate(tek);

  while(redo) {

  /*TO_INDICATE_WORKING_AREA*/
  fillarea(4, pr);

  more = TRUE;

    while(more) {

    /*SET_CHARACTER_HIGHT*/
    init_val(tek, 1, &ht, 1, &drct1, &drco1);
    req_val(tek, 1, &ht);
    s_ch_ht(ht * 0.5);

    /*SET_CHARACTER_EXPANSION_FACTOR_WITH_BUNDLE*/
    init_val(tek, 1, &ef, 1, &drct2, &drco2);
    req_val(tek, 1, &ef);
    tx->tx_chef = ef * 2.0;
    tx->tx_fp.fp_fo = 1; /*TO_ESCAPE_ERROR*/
    tx_rep(tek, 3, tx);

    /*INPUT_STARTING_POINT*/
    init_loc(tek, 1, &loc, 1, &drct3, &drco3);
    req_loc(tek, 1, &loc);

    /*SET_CHARACTER_UP_VECTOR*/
    init_val(tek, 1, &vx, 3, &drct4, &drco4);
    req_val(tek, 1, &vx);
    v->w_x = vx;
    init_val(tek, 1, &vy, 3, &drct5, &drco5);
    req_val(tek, 1, &vy);
    v->w_y = vy;
    s_ch_up(v);

    /*SET_TEXT_FONT_AND_PRECISION*/
    init_cho(tek, 1, &cho2, 3, &drct6, &drco6);
    req_cho(tek, 1, &cho2);
    fopr.fp_fo = (Tfont)(cho2 +1);
    init_cho(tek, 1, &cho1, 3, &drct7, &drco7);
    req_cho(tek, 1, &cho1);
    fopr.fp_pr = (Tprec)(cho1 - 1);
    s_tx_fp(fopr);

    /*INPUT_STRING*/
    init_str(tek, 1, str, 1, &drct8, &drco8);
    req_str(tek, 1, str);

    /*OUTPUT_TEXT*/
    text(str, &loc.loc_pt);

    /*CHOICE_CONTINUE_OR_STOP*/
    init_cho(tek, 1, &cho3, 3, &drct9, &drco9);
    req_cho(tek, 1, &cho3);
    if (cho3 == 1) redo = FALSE;
    if (cho3 != 3) more = FALSE;
    if (redo && !more) clear(tek);
    }
  }

  /*DEACTIVATE_WORKSTATION*/
  deactivate(tek);

  /*CLOSE_WORKSTATION*/
  close_ws(tek);

  /*CLOSE_GKS*/
  close_gks();
}
```

このプログラムを実行した結果が、次ページの図４－１である。インタラクションの経過を追って多数の図を示したいのではあるが、紙面の都合上これだけにとどめる。

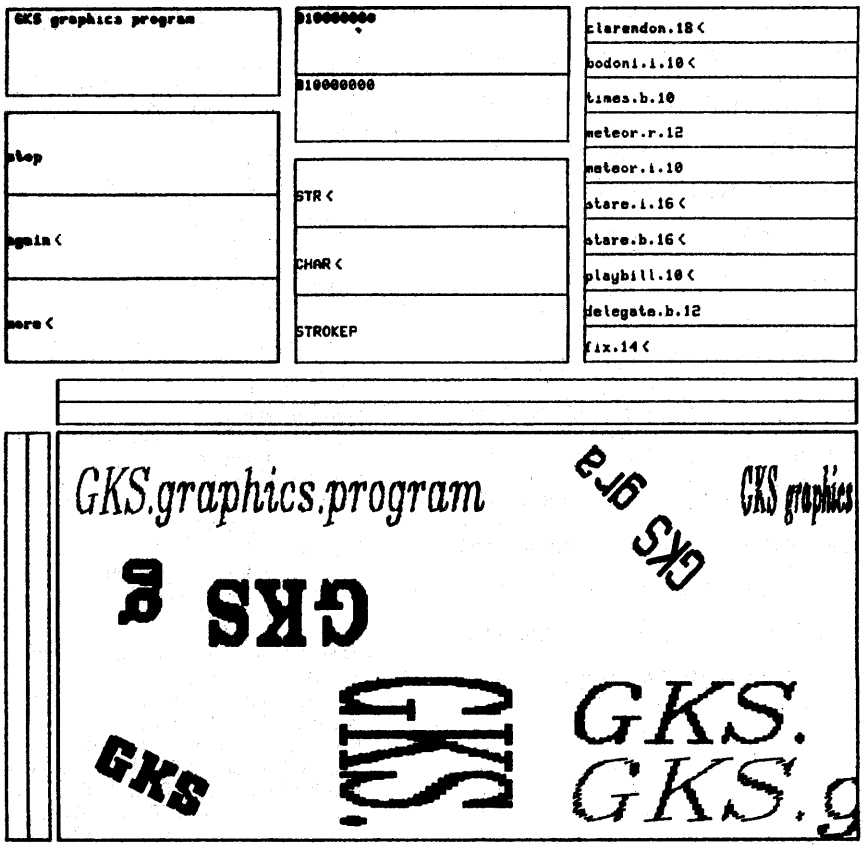

図４－１　　実行例

５。　ＧＫＳ利用のための環境と今後

ＧＫＳではグラフィクス・デバイスに独立に、しかもその機能を効率よく利用するために、ワークステーションという概念を持ち込んでいる。しかもこのワークステーションが大きな役割を担っている。前節までで述べてきたように、プログラムを書くにあたって、workstation dependent な相とworkstation independent な相の２つの相を持っていることが常に意識される。そして具体的に挙げてみても、座標変換のシステム、出力命令における属性指定の方法、入力命令を扱う論理入力デバイス、セグメントの管理と、いかなる場合においてもワークステーションの機能は非常に重要なものになっている。つまりＧＫＳでは、ワークステーションの機能を定義し、ＧＫＳ全体の機能はこれを前提にして実現されているわけであり、ワークステーション機能の実現がＧＫＳ利用のための条件といえる。

既存のグラフィクス・デバイスでは、このワークステーションの機能をすべてサポートしているものはない。そこで、現時点ではＧＫＳ・プログラムパッケージ内でワークステーションのシミュレーションを行ない、デバイスの機能を補うことになる。具体的に言うと、ＬＯＣＡＴＯＲに相当する機能しか持たないデバイスで、その表示面の操作などによって、ＣＨＯＩＣＥと同等の機能を実現したりするわけである。４節のプログラム例で用いているＣＨＯＩＣＥ機能は、このようにして実現されたものである。

しかしこの手法では、効率性などの面で様々な問題を抱えている。出力機能の点では、Output primitiveの属性指定でそれが見受けられる。特に text fontの場合に顕著である。

4節のプログラムで利用した font は、ＧＫＳ・プログラムパッケージが UNIX にのっている font fileから TEKTRONIX 4014端末のコードを生成し、これを送ることで表示するに至っている。従って表示するまでに、fileの読み取りと TEKTRONIX4014端末のコード生成と転送という手間がかかっている。実際に１個の文字を表示するのに、 9600 ボーの回線を用いても１０秒以上かかってしまうこともある。もしデバイスが font を管理できれば、ＧＫＳ・プログラムパッケージは文字のコードを送るだけでよくなる。入力機能についてもまったく同じことで、シミュレーションに要する手間はかなりのものとなる。このようにＧＫＳ・プログラムパッケージによるシミュレーションは、決して理想的なものではないと思われる。以上のことを踏まえると、今後ＧＫＳ本来の機能を効率よく利用できるようになるために、個々のグラフィクス・デバイスそのものがＧＫＳのワークステーションの機能を持つことが肝心であり、その早急な実現が望まれよう。

【 参考文献 】

[1] Information Processing Graphical Kernel System Functional Description, Draft International Standard ISO/DIS 7942, 1982.11.

[2] G.Enderle,K.Kansy,G.Pfaff: Computer Graphics Programming, Springer-Verlag,1984.

[3] F.R.A.Hopgood et. al.: Introduction to The Graphical Kernel System (GKS), Academic Press, 1983.

[4] 穂坂 衛： ISO グラフィックス標準，グラフィックスとCAD 研究会， 1983.3.

[5] 木村文彦： ISO グラフィックス標準 GKS, ibid.

《 付録 》 C-bindng のＧＫＳでの型宣言の一部

```
typedef char *String;
typedef enum { FALSE, TRUE } Bool;
typedef unsigned Int;
typedef float Real;
typedef struct {
        Real    d_x, d_y;
        } Dc;
typedef struct {
        Dc      d_ll, d_ur
        } Drect;
typedef struct {
        Real    w_x, w_y;
        } Wc;
typedef struct {
        Real    n_x, n_y;
        } Nc;
typedef struct {
        Wc      w_ll, w_ur;
        } Wrect;
typedef struct {
        Nc      n_ll, n_ur;
        } Nrect;
typedef struct {
        Wrect   nt_wind;
        Nrect   nt_view;
        } Ntran;

/******text representations******/
typedef Real Charef;
typedef Real Charht;
typedef Real Charsp;
typedef Int Tfont; /* not a String"?*/
typedef enum { STR, CHAR, STROKEP } Tprec;
typedef enum { RIGHT, LEFT, UP, DOWN } Path;
typedef enum { HNORMAL, ALEFT,
               CENTRE, ARIGHT } Talhor;
typedef enum { VNORNAL, TOP, CAP,
               HALF, BASE, BOTTOM } Talver;
typedef struct {
        Talhor  ta_hor;
        Talver  ta_ver;
        } Talign;
typedef struct {
        Tfont fp_fo;
        Tprec fp_pr;
        } FoPr;
typedef struct {
        Bindex  tx_ix;
        FoPr    tx_fp;
        Charef  tx_chef;
        Charsp  tx_chsp;
        Cindex  tx_ci;
        } TextRep;

/******fillarea representations******/
typedef Int Sindex;
typedef enum { HOLLOW, SOLID,
               PATTERN, HATCH } Istyle;
typedef struct {
        Bindex  fa_ix;
        Istyle  fa_is;
        Sindex  fa_si;
        Cindex  fa_ci;
        } AreaRep;
```