Qüestió - V. 9, n.º 1 (març 1985) pp. 35-53

UNA IMPLEMENTACIÓ REAL DE L'ESTANDARD GRÀFIC GKS

J. ARESPACOCHAGA*, R. JUAN*, E. TORRES**, M. D. VICENTE*
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abans de l'any 1982 no s'havia acceptat cap stàndard gràfic. ISO el 1982 i ANSI el 1984 acceptaren el Graphical Kernel System (GKS). Aquest paper intenta donar unes idees generals sobre una implementació real d'aquest stàndard. Sobretot hem aprofundit més en els temes que han quedad més foscos a l'estàndard com són segments, metafile, i el concepte de multiestació.

Keywords: GKS, GRAPHIC STANDARD, DRAWING PRIMITIVES, COORDINATES TRANSFORMATION WORKSTATION

1.- INTRODUCCIO.

Fa molt de temps que la comunitat d'informàtics que treballen a l'àrea gràfica esperaven un estàndard que permetés d'unificar tant el vocabulari com la interpretació dels conceptes usualment manegats així com la portabilitat dels programes amb la màxima independència possible dels estris hardware utilitzats.

Mentre que en els llenguatges de programació aparegueren molt aviat estàndards "de facto" (FORTRAN, ALGOL,...) la qual cosa portà ràpidament als estàndards internacionals, en l'àrea gràfica l'existència d'estàndards regionals des de fa alguns anys, no havia provocat l'aparició de cap estàndard internacionalment conegut.

Ara, després de més de 6 anys de treball portat a terme per un grup internacional d'experts en gràfics, aquesta situació ha estat superada amb l'acceptació del Grafical Kernel System (GKS) com a estàndard internacional per part de l'organització ISO el 1982.

GKS és un sistema gràfic el qual permet que els programes suportin una gran varietat d'aparells gràfics, havent estat definit d'una forma totalment independent de qualsevol llenguatge de programació.

El present treball té com a objectiu presentar una implementació concreta de l'estàndard, realitzada amb PASCAL i FORTRAN com a llenguatges de programació, tot explicant les solucions emprades davant dels problemes derivats de les possibles indefinicions i contradiccions del propi estàndard.

2.- PRINCIPALS CONCEPTES GKS.

Els principals conceptes d'un sistema gràfic estan íntimament lligats a les tasques que ha de desenvolupar. Al cas que ens ocupa, aquests conceptes són els següents:

1. Sortida. Lligat a la tasca de generació de dibuixos. Correspon a la sortida gràfica.

2. Sistema i transformació de coordenades. Els dibuixos són creats per l'usuari en un o més d'un sistemes de coordenades.

- *J.Arespacochaga, *R.Juan, *M.D.Vicente - Centre de Càlcul de la Universitat Politècnica de Catalunya
Av. Gregorio Marañón, s/n. Barcelona.

-**E. Torres - E.T.S.E.I.B. Dep. Mètodes Informàtics de la Universitat Politècnica de Catalunya
Av. Diagonal, 647 - Barcelona.

Aquests dibuixos poden ser representats sobre diferents perifèrics gràfics amb diferents sistemes de coordenades.

3. Estació de treball. Unitat constituïda per una superfície de representació (display, plotter,...) i un element d'entrada (teclat, tauleta,...). Aquest concepte, també conegut amb el mot anglès "workstation" és una contribució del GKS a la metodologia del disseny de sistemes gràfics.

4. Entrada. Lligat a la tasca de comunicació de l'estació de treball amb l'operador . Correspon a l'adquisició tant d'informació geomètrica com alfanumèrica i identificació d'elements gràfics.

5. Segment. Conjunt d'elements gràfics que poden ser manipulats independentment de la resta. El GKS contempla l'emmagatzematge de segments en forma independent de les estacions de treball més els procediments destinats a copiar segments sobre les estacions.

6. Metafile. És una forma d'emmagatzemar dibuixos amb propòsit d'arxiu i d'intercomunicació entre diferents sistemes gràfics.

7. Llistes d'estat. A cada instant, el GKS s'hi troba en un determinat estat el qual es representa mitjançant uns valors emmagatzemats en vàries llistes. Aquests valors els modificaran les funcions GKS utilitzades pels programes d'aplicació.

8. Nivells. Les funcions GKS s'agrupen en 9 nivells diferents segons les facilitats d'entrada/sortida implementades tal com es representa a la figura 1.

	NIVELL ENTRADA		
	A	B	C
NIVELL SORTIDA			
0	No hi ha entrada. Control minim. Atributs predefinits.	Entrada en Request. Seleccio modus. Inicialitzacio de periferic. No hi ha Pick.	Entrada en Sample i Event. No hi ha Pick.
1	Sortides completes. Atributs redefinibles. Segments i metafile.	Request Pick. Seleccio modus. Inicialitzacio del Pick	Entrada en Sample i Event pel Pick.
2	Tot el precedent mes segments emmagatzemats independentment de l'estacio.	Tot el precedent.	Tot el precedent.

Figura 1. Nivells GKS

9. Errors. El GKS defineix a més de les funcions, unes determinades condicions d'error que poden produir-se en processar una funció la qual cosa permet que el programa d'aplicació endegui les accions que cregui adients.

10. Dimensionalitat. El GKS és un sistema de dues dimensions. Això vol dir que totes les entrades/sortides de coordenades són bidimensionals. Actualment s'està estudiant la definició de l'ampliació GKS a tres dimensions.

3.- INTERFICIES DEL GKS.

Malgrat la independència de la definició de les funcions GKS del llenguatge de programació, en una implementació del sistema, aques tes funcions cal materialitzar-les com a sub rutines o procediments en un determinat llen guatge. A més, els programes d'aplicació han de poder usar les funcions GKS i tot això ha d'estar suportat pel sistema operatiu.

El model de capes representant a la figura 2 il.lustra el paper del GKS en un sistema grà fic on, totes les funcions gràfiques són realitzades únicament via les funcions GKS.

sinó d'estació.

La WDT pot ser consultada pel programa d'apli cació però no modificada.

La llista d'estat del GKS (GKS State List) - que conté les variables globals del GKS en un moment donat de l'execució del programa. Són 'valors actuals' que poden ser consultats i modificats pel programa d'aplicació. Són variables globals dels GKS i per tant corresponen a una estació de treball en particular.

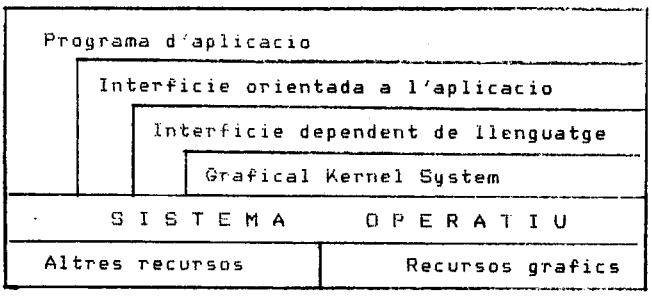


Figura 2. Model de capes de sistema grafic

4. ESTRUCTURACIO DE LES DADES.

El GKS defineix dos tipus fonamentals d'estructures de dades:

1. Taules de descriptors.
2. Llistes d'estats.

La taula descriptora del GKS (GKS Description Table) on es defineixen característiques generals de la implementació i que poden ser consultades per l'aplicació.

La taula descriptora de cada tipus d'estació de treball (Workstation Description Table) on la implementació GKS descriu les característiques de tots els tipus d'estacions de què disposi en la implementació.

Així dues o més estacions del mateix tipus - queden definides a una única WDT.

A la WDT es descriuen aquelles capacitats de l'estació directament relacionada amb el GKS, no és una taula descriptora de dispositiu

Quan el programa d'aplicació vol accedir a una estació de treball via la funció OPEN WORKSTATION, el GKS crea una llista d'estat de l'estació (Workstation State List) que conté totes les característiques de l'estació que poden ser canviades directament o indirecta pel programa d'aplicació.

La WSL és inicialitzada a partir de la WDT corresponent al tipus d'estació especificat a la funció d'apertura de -- l'estació.

Una de les entrades de la GKSSL és una nova estructura de dades, la llista d'estat de ca da segment (Segment State List).

5.- IMPLEMENTACIO DE LES RUTINES DE GESTIO DE SEGMENTS.

5.1 INTRODUCCIÓ.

L'estandard gràfic GKS estableix la possibi-

litat d'agrupar les primitives de dibuix en unitats anomenades 'segments'. Aquestes unitats permeten d'identificar parts d'un gràfic per llavors aplicar les funcions de transformació gràfica. Aquestes funcions també vénen establertes en l'estàndard.

Els segments están agrupats en una estructura d'ordre superior anomenada 'display file'. El GKS només estableix quina informació ha de contenir el display file, i quines funcions cal que suporti una implementació del mateix; però no defineix l'estructura concreta que han de tenir les dades, ni el seu format. Veiem con s'han resolt aquests punts en la nostra implementació.

5.2 ESTUDI PREVI.

Els segments estan formats per dos tipus d'informació ben definits:

1. La 'capçalera', que conté informació relativa al segment pròpiament dit com són el nom i els atributs. Aquesta informació és quantificable ja que està perfectament definida.

2. El 'cos' del segment, que conté les primitives de dibuix associades. Aquesta informació està totalment indefinida, ja que pot variar en dos sentits: El nombre de primitives per segment i el nombre de punts per primitiva. A més tampoc està definit el format d'aquestes dades.

Aquesta dualitat suggereix una divisió física de la informació en dues estructures diferents: per una banda les capçaleres, formant una llista en un àrea fixa de memòria i amb una estructura definida, i per una altra una zona de memòria del més gran tamany possible, segmentada en blocs de longitud variable (un per a cada capçalera) que contenen les primitives de dibuix. A partir d'ara ens referirem a les dues àrees com 'display file' (tractant-se de la llista de capçaleres) i 'display program' (quan es tracta dels cossos de segments).

Per altra part, les funcions que es realitzaran amb més freqüència amb els segments són les de cerca (pel nom) i el dibuix del conjunt de segments segons la seva prioritat (un dels atributs associats a cada segment). Per a aconseguir això amb la màxima velocitat, el display file s'organitza com una llista doblement enllaçada: pel nom (amb els segments ordenats alfabèticament) i per prioritat (amb els segments ordenats de major a menor prioritat). Això obliga a que la resta de funcions de gestió dels segments (creació o esborrat) així com les que modifiquen el nom o la prioritat mantinguin aquesta ordenació.

Tant les capçaleres del display file com els blocs de memòria del display program poden tenir dos status: 'buit' o 'ple'. Inicialment totes les capçaleres estan 'buides' car no existeix cap segment definit. Cada cop que es crei un nou segment caldrà agafar una d'aquestes capçaleres 'buides', les quals estan organitzades com una pila enllaçada.

En quant als blocs de memòria, cada cop que s'esborra un segment, el seu bloc associat passa al status de 'buit', ocupant una zona de memòria que ha de ser aprofitada per successius segments de la forma més eficient possible. La gestió d'aquests blocs es realitza considerant-los com a segments buits organitzats com una llista doblement enllaçada.

Resumint, hi ha tres tipus de capçaleres en el display file:

1. Capçaleres buides (o no utilitzades) formant una pila.

2. Capçaleres plenes que contenen un segment, formant dues llistes enllaçades: per nom i per prioritat.

3. Capçaleres plenes que contenen un bloc lliure de memòria i formen també dues llistes: per adreça i per tamany.

La necessitat de mantenir enllaçades les capçaleres 'segment' per nom i per prioritat ja ha estat justificada anteriorment. En quant als blocs lliures, es mantenen enllaçats per longitud perquè en crear un nou segment (ja què a priori no pot saber-se quina longitud tindrà) se l'hi assigna el bloc lliure de major tamany, i l'enllaç per adreça serà de gran utilitat quan la fragmentació del display program obligui a fer un compactació.

5.3 ESTRUCTURES DE DADES.

El comportament del display file es simula amb estructures del tipus ARRAY:

```
TYPE
vis   = (visible, invisible)          ;
hig   = (highlighted, normal)         ;
det   = (detectable, indetectable)    ;
  capçalera= RECORD
       nom                : CHAR    ;
       adreça             : INTEGER ;
       longitud           : INTEGER ;
       prioritat          : REAL    ;
       visibilitat        : vis     ;
       highlighting       : hig     ;
       detectabilitat     : det     ;
       enllaç_pl          : INTEGER ;
       enllaç_nd          : INTEGER
      END ;
 VAR dfile   : ARRAY  -2.. maxsegments   OF
capçalera ;
   d_program : ARRAY  1..total_memoria
                                 OF INTEGER;
     stack_pointer_capçaleres:     INTEGER;
```

El tipus capçalera és un enregistrament els camps del qual signifiquen:

1. nom. Nom del segment (Un identificador).

2. adreça. Adreça física de memòria on comença el bloc que conté el cos del segment.

3. longitud. Longitud del bloc.

4. prioritat, visibilitat, highlighting, detectabilitat. Atributs del segment.

5. enllaç_pl. Quan la capçalera pertany a un segment, aquest serveix per a enllaçar per prioritat (p); si pertany a un bloc lliure, serveix per a enllaçar per longitud (l).

6. enllaç_nd. Quan la capçalera pertany a un segment, aquest serveix per a enllaçar per nom (n); si pertany a un bloc lliure, serveix per a enllaçar per adreça (d).

El display file és un ARRAY de capçaleres -- 'vàlides' (de l'1 a un màxim que depèn de la implementació) al que s'afegeixen tres capçaleres 'fictícies' (del -2 al 0) amb funcions auxiliars, que contenen els apuntadors als primers elements de cada llista i serveixen per a simplificar els algorismes d'inserció, esborrat, etc...

5.4 COMPACTACIÓ (GARBAGE COLLECTION)

Durant l'execució del programa de gestió del display file s'observa que el display program té una forta tendència a fragmentar-se en blocs cada cop més petits, el que impedeix un aprofitament eficient de la memòria disponible. Davant aquest fet, és necessari compactar els segments en memòria, desplaçant-los cap a les posicions més baixes de memòria, de forma que s'obtingui al final un únic bloc de memòria lliure darrera dels segments.

Estat inicial:

```
-------------------------------------------
11122AAAAAABBBBBB333CCCCCC44445555DDDDDD6666
-------------------------------------------
```

estat final:

```
-------------------------------------------
AAAAAABBBBBBCCCCCCDDDDDD6666666666666666666
-------------------------------------------
```

La figura representa els estats inicial i final del display program: on les lletres signifiquen diferents segments, i els nombres diferents blocs de memòria lliure.

La compactació té a més la propietat d'alliberar capçaleres ocupades per blocs de memòria lliure. En efecte, en l'exemple anterior, es precisaven sis capçaleres (una per cada bloc lliure) en la situació inicial, mentre que després de la compactació només es precisa una; les altres es posen en la pila i resten disponibles per a futurs segments.

S'han estudiat dues situacions límit en les que cal compactar. La primera quan es buida la pila de capçaleres lliures (com hem vist, la compactació allibera capçaleres). La segona quan durant la creació d'un segment es -- precisa més espai del que se l'hi ha assignat al obrir-lo.

En el segon cas hi ha dues alternatives, que estaran igualment disponibles per a l'usuari:

1. La primera (i més senzilla) consisteix en tancar el segment, de forma que l'usuari perdi la informació restant; queden sota la seva responsabilitat l'esborrar altres segments per aconseguir espai - lliure.

2. La segona consisteix en tancar provisionalment el segment, fer una compactació, copiar sobre un nou bloc amb més espai el contingut de l'antic deixant-lo obert per seguir treballant, i finalment, esborrar l'antic segment.

La segona alternativa no garanteix que s'aconsegeixi sempre més espai però dóna suport a l'usuari, al qual, en darrer terme, sempre l'hi queda el recurs d'aplicar la primera.

6. UBICACIO PUNTS GEOMETRICS:

A una estació de treball GKS, la major part de les funcions de dibuix, Polyline, Polymarker, Fillarea resten definides geomètricament mitjançant dues variables, el nombre de punts i les coordenades dels punts. En PASCAL seria

```
TYPE
 point_ndc = ARRAY [1..2] OF REAL;

 point_nc_array = RECORD

 n_punts : INTEGER;
 punts   : ARRAY [1..maximpunts] OF point_ndc
 END
```

La modularització dels processos a realitzar a l'estació de treball resta facilitada amb la utilització de dues variables d'aquest tipus que anomenem punts_in i punts_out.

Veiem els processos en detall

a) La variable 'punts_in' pot inicialitzar-se de dues formes:

   1. Carregar directament la primitiva tal com arriba.

   2. Carregar el procés de 'redibuix' a partir del 'Display File'.

      punts_in = Segment_Transformation (punts_in)

b) Procés de clipping

   punts_out = Clipping (punts_out)

Una estació de treball amb els processos ben modularitzats facilita el manteniment del software de l'estació i a més pot permetre que algún d'ells sigui realitzat per elements hardware.

7. EL GKS MULTIESTACIO.

El GKS està dissenyat per poder gestionar simultàniament vàries estacions de treball aprofitant al màxim llurs capacitats.

El concepte abstracte d'estació de treball és el conjunt format per un dispositiu físic amb capacitat gràfica (terminal) i el software necessari que li doni la capacitat funcional que exigeix l'estàndard.

A un sistema GKS totes les estacions han de tenir la mateixa funcionalitat definida pel nivell d'implementació. És a dir, el nivell d'implementació es refereix a totes les estacions, no pot haver una amb una més gran o més petita capacitat funcional.

La materialització d'una estació de treball es realitza mitjançant el terminal gràfic, el problema consisteix en com donar una mateixa capacitat funcional a dispositius amb distinta capacitat gràfica.

L'estructura de dades de l'estàndard indica explícitament la possibilitat multiestació amb l'existència de la llista de diferents tipus d'estacions de treball. D'entrada se'ns planteja la necessitat d'un manegador de les estacions de treball per a anar distribuint les diferents comandes, generades pel programa d'aplicació via GKS, per a què arribin -- fins a les estacions de treball que en terminologia GKS estiguin obertes.

D'altra banda, l'existència d'aquesta llista de diferents tipus d'estació significa que hom podrà connectar estacions de treball amb diferents nivells d'intel.ligència. Se'ns -- planteja així la necessitat de muntar dife-- rents drivers per tal d'aconseguir que l'arribada d'una comanda de certa complexitat pugui ser executada pel terminal. El concepte de -- driver en aquest cas és el d'esmicolar una

comanda complexa en vàries comandes assequibles, és a dir, intel.ligibles per a l'estació de treball.

La figura 3 representa l'arquitectura del sistema

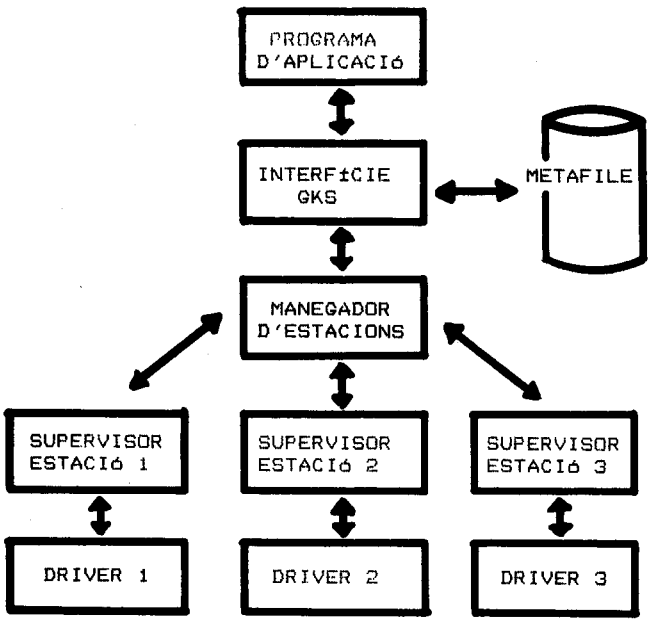


Fig. 3: Arquitectura GKS multiestació.

Un cop decidida l'estructura del sistema en varis mòduls especialitzats cal decidir com es comunicaran els diferents nivells.

La comunicació des del programa d'aplicació cap al GKS és a nivell de crida de les rutines que té el GKS.

La comunicació des del GKS cap al manegador d'estacions de treball ha estat implementada mitjançant un buffer per a fer arribar les comandes cap al manegador d'estacions de treball, i que alhora faci arribar al GKS paràmetres d'entrada demanats pel GKS i que s'han entrat a l'estació de treball amb els dispositius d'entrada.

La comunicació entre el manegador d'estacions de treball i els drivers de les estacions és a nivell de crida de les rutines pròpies de cada estació.

Un problema addicional en la implementació d'un GKS nivell 1b, és el sistema d'emmagatzemar els segments, i que el GKS anomena 'Device Dependent Segment Storage' (DDSS). Si un sistema disposa de terminals amb capacitat pròpia de gestió, haurà d'ésser realitzada pel terminal i no pas pel driver, no està definida però, quina és l'acció que ha de fer el driver d'un terminal sense capacitat de gestió de segments. Hi ha dues alternatives:

1. Que cada driver gestioni un sistema d'emmagatzemar segments per a cada estació de treball.

2. No realitzar cap mena d'acció.

La primera alternativa és molt costosa en recursos del sistema car les necessitats de memòria poden ser molt importants. Un sistema que adopti aquesta solució manté la mateixa informació a diferents llocs, al programa d'aplicació i als drivers de cada una de les estacions. Malgrat tot, aquesta és la solució més d'acord amb la filosofia del GKS que no contempla la possibilitat de disposar en un sistema d'estacions de treball amb diferent capacitat funcional.

L'alternativa 'cap mena d'acció' pot crear problemes de corrupció de les estructures de dades si no es prenen les mesures corresponents.

Com veieu cap de les dues alternatives pot ser considerada com a veritable solució del problema. Les solucions a llarg termini poden estar a:

1. Modificar les estructures de dades del GKS per a què es puguin gestionar estacions de treball amb diferent capacitat funcional, associant el nivell o subnivell d'implementació a l'estació de treball i no al sistema.

2. Esperar que l'actual tendència en dotar d'una més gran capacitat gràfica als terminals porti una estandardització funcional dels terminals i una definició funcional de comunicació entre el sistema GKS i l'estació de treball.

8. EL GKS METAFILE.

El GKS disposa d'un mecanisme per a l'emmagatzemament indefinit en cinta o disc de la informació, gràfica i no gràfica generada per tot programa d'aplicació.

Un cop emmagatzemada aquesta informació, les sortides gràfiques del programa poden ser reproduïdes a qualsevol lloc i moment, únicament amb la lectura i interpretació del fitxer que ha estat gravat. Aquesta interfície s'anomena GKS METAFILE (GKSM) i gràcies a ella es fa possible:

1. Transportar informació gràfica entre diferents sistemes.

2. Transportar informació gràfica entre diferents ordinadors.

3. Transportar informació gràfica entre diferents programes d'aplicació.

4. Emmagatzemar i transportar informació auxiliar no gràfica.

En la utilització del Metafile s'estableixen dues directives de treball: una de programa GKS a Metafile (escriptura de Metafile) i una altra de Metafile a programa GKS (lectura del Metafile), com s'esquematitza a la figura 4.

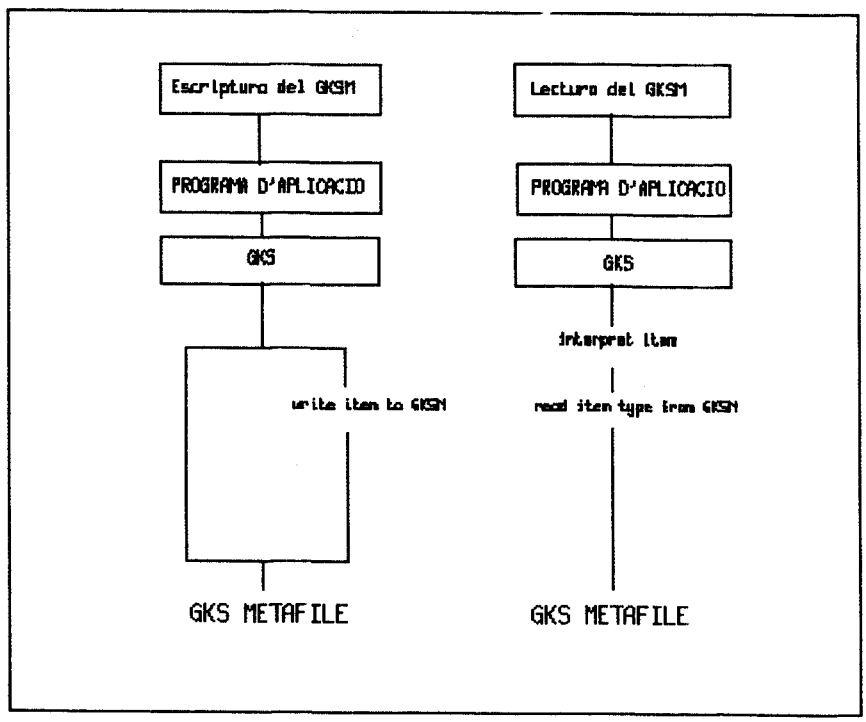


figura 4 Lectura/Escriptura del Metafile

Tant en l'escriptura com en la lectura del Metafile, el GKS disposa d'un conjunt de rutines específiques. La crida, dins del programa d'aplicació a una rutina GKS que hagi de quedar reflectida en el fitxer del Metafile es tradueix en l'escriptura d'un enregistrament en ell.

La gravació dels enregistraments es realitza automàticament, sense que calgui fer cap crida a les rutines GKS relatives al Metafile. El programa d'aplicació té, a més, la possibilitat de gravar el Metafile enregistraments amb contingut d'informació nö gràfica utilitzant directament la rutina WRITE_ITEM_TO_GKSM.

A l'hora de llegir i interpretar el contingut del fitxer, el procés seguit és el seguent, cada enregistrament és llegit del fitxer Metafile; tot seguit s'obté el tipus d'aquest enregistrament i, en funció d'aquest tipus, s'interpreta el contingut de l'enregistrament reproduint-se l'execució de la rutina GKS que en el seu moment s'emmagatzema a l'arxiu Metafile.

Malgrat que el document de definició de l'estàndard GKS no dóna cap tipus de Metafile estandaritzat en quant al seu contingut i format, s'ha optat en aquesta implementació per seguir en tot el possible el suggerit per aquest document.

Des del moment en què és activada una estació de treball Metafile Output, el GKS envia els paràmeters de les diferents crides a rutines GKS, a més de al manegador d'estacions, a un altra rutina que s'encarregarà de codificar i escriure els enregistraments que composaran l'arxiu Metafile.

Aquesta rutina codifica els paràmetres que - rep segons el format que correspon al seu tipus i l'escriu en un buffer que va omplint.

Cada registre que s'escrigui en el fitxer Metafile correspondrà a una crida a una rutina GKS, i començarà amb:

1. Escriptura de la capçalera d'enregistrament en un buffer.

Per a cadascun dels paràmetres que arriben, es realitza:

1. Codificació segons el format que correspon al seu tipus.

2. Escriptura del paràmetre, ja codificat, en el buffer.

Un cop han arribat tots els paràmetres de la rutina només cal

1. Codificar i escriure en el buffer la llargada d'enregistrament, la qual ha estat calculada mitjançant un comptador durant tot el procés de tractament de paràmetres.

Amb això, el buffer resta definit pel conjunt {capçalera de registre codi d'operació llargada de registre paràmetres de la rutina}. Tot seguit hom buida el buffer en el fitxer Metafile contingut en cinta o disc, en el que constitueix un nou enregistrament.

El procés de lectura i interpretació del contingut del fitxer comença amb la lectura d'un enregistrament de l'arxiu Metafile i acaba - amb la crida a la rutina GKS que reprodueix la que va fer el programa que va generar l'escriptura del Metafile:

1. Lectura d'un enregistrament de l'arxiu a cinta o disc.

2. Emmagatzemament de l'enregistrament en un vector de caràcters.

3. Decodificació dels sis bytes corresponents al codi d'operació.

4. Decodificació, en funció del codi d'operació, de la resta de paràmetres de la rutina GKS.

5. Crida de la rutina GKS.

Cada rutina GKS té un i només un codi d'operació, de manera que en ser decodificat - resten perfectament determinats els paràmetres que completen el contingut de l'enregistrament.

Les rutines de més baix nivell, tant del procés de lectura com del d'escriptura, relacionades amb la codificació, confecció, escriptura, lectura i decodificació dels enregistraments han estat implementades en FORTRAN,

donada la més gran facilitat d'aquest llenguatge envers el PASCAL a l'hora de tractar informació de dimensions desconegudes a priori a causa de la dependencia de la llargada dels paràmetres enviats per cada rutina. La utilització del FORTRAN fa possible que l'ocupació de memòria pel Metafile sigui l'estrictament necessària per a l'emmagatzemament d'informació.

Donat que l'estandard GKS no especifica res respecte els noms dels fitxers que s'escriuen i llegeixen amb el Metafile, i per analogia amb molts sistemes que utilitzen el llenguatge FORTRAN, s'ha optat en aquesta implementació d'anomenar els fitxers -- creats i llegits pel Metafile amb un nom de sis caràcters "GKSMnn"a on "n" és el nombre identificador del connector amb el que ha estat oberta l'estació Metafile. D'aquesta manera, hi ha la possibilitat de crear fins a cent fitxers [GKSMOO ... GKSM99].

S'ha observat en la filosofia del Metafile, que s'esmenta en el document de l'estandard GKS, un aspecte que pot ocasionar certs problemes a l'hora d'utilitzar aquesta interfície: els fitxers creats pel Metafile emmagatzemen únicament les crides a subrutines GKS que es produeixen des del moment que s'activa l'estació de treball Metafile fins que és desactivada. El Metafile no conté, doncs, res del que s'hagi pogut produir, pel que respecta a modificacions dels atributs de representació, al programa d'aplicació abans de l'activació del Metafile.

Com a conseqüència d'aixo quan hom vulgui reproduir el que hi ha emmagatzemat al Metafile, les primitives de dibuix s'aniran dibuixant amb els atributs de representació vigents en el programa d'aplicació que llegeix el Metafile, i no amb els que hi havia en el programa que va generar el fitxer; això succeirà mentre no es reprodueixin les possibles assignacions d'atributs que s'havien fet al programa d'aplicació que va omplir el Metafile. Per tant, els gràfics resultants de la lectura i interpretació d'un fitxer Metafile poden tenir una apariència del tot diferent a la del generat pel programa original.

Com a possible solució a aquest problema hom veu la necesitat d'emmagatzemar en el fitxer, al moment de l'activació de l'estació Metafile Dutput, els valors de tots els atributs de representació vigents en el programa d'aplicació, per a què siguin automàticament carregats en les corresponents estructures de dades del programa d'interpretació del Metafile.

9. ANNEX DE LES PRINCIPALS PRIMITIVES GRAFIQUES.

9.1 NIVELL Oa

ACTIVATE WORKSTATION
CELL ARRAY
CLEAR WORKSTATION
CLOSE GKS
CLOSE WORKSTATION
DEACTIVATE WORKSTATION
FILL AREA
GENERALIZED DRAWING PRIMITIVE (GDP)
OPEN GKS
OPEN WORKSTATION
POLYLINE
POLYMARKER
SELECT NORMALIZATION TRANSFORMATION
SET CHARACTER HEIGHT
SET CHARACTER UP VECTOR
SET CLIPPING INDICATOR
SET COLOUR REPRESENTATION
SET FILL AREA INDEX
SET POLYLINE INDEX
SET POLYMARKER INDEX
SET TEXT INDEX
SET VIEWPORT
SET WINDOW
SET WORKSTATION VIEWPORT
SET WORKSTATION WINDOW
TEXT (GTX)

9.2. NIVELL Ob

INITIALISE CHOICE
INITIALISE LOACTOR
INITIALISE STRING
INITIALISE VALUATOR
REQUEST CHOICE
REQUEST LOCATOR
REQUEST STRING
REQUEST VALUATOR
SET CHOICE MODE
SET LOCATOR MODE
SET STRING MODE
SET VALUATOR MODE
SET VIEWPORT INPUT PRIORITY

9.3. NIVELL 1a

ACCUMULATE TRANSFORMATION MATRIX
CLOSE SEGMENT
CREATE SEGMENT
DELETE SEGMENT
DELETE SEGMENT FROM WORKSTATION
EVALUATE TRANSFORMATION MATRIX
REDRAW ALL SEGMENTS ON WORKSTATION
RENAME SEGMENT
SET CHARACTER EXPANSION FACTOR
SET CHARACTER PATH
SET CHARACTER SPACING
SET FILL AREA REPRESENTATION
SET HIGHLIGHTING
SET PATTERN REFERENCE POINT
SET PATTERN REPRESENTATION
SET PATTERN SIZE
SET POLYLINE REPRESENTATION
SET POLYMARKER REPRESENTATION
SET SEGMENT PRIORITY
SET SEGMENT TRANSFORMATION
SET TEXT REPRESENTATION
SET VISIBILITY

9.4. NIVELL 1b

INITIALISE PICK
REQUEST PICK
SET DETECTABILITY
SET PICK IDENTIFIER
SET PICK MODE

10. EXEMPLES DE SORTIDA DE LES PRIMITIVES D'OTPUT.

S'inclouen seguidament una sèrie de dibuixos obtinguts mitjançant el GKS que són representatius de les primitives bàsiques de dibuix.

En la figura A es veuen 4 "polyline" de 7 punts, s'ha utilitzat per a cada una un tipus de línia diferent.

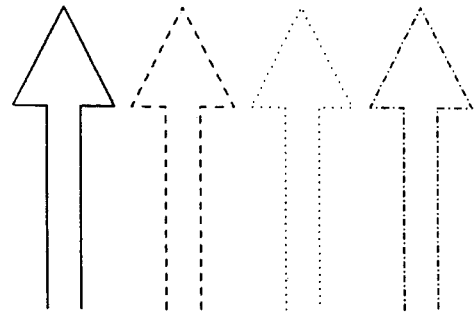


Fig. A: Primitiva "Polyline"

En la figura B s'han utilitzat els mateixos punts que a la figura 1 però amb la primitiva "polymarker".

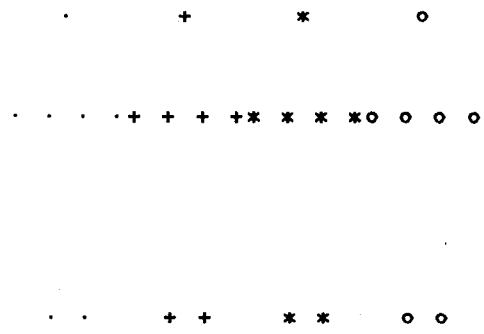


Fig. B: Primitiva "Polymarker"

La figura C és un exemple de la utilització de la primitiva "text" i els seus atributs, s'ha dibuixat 8 cops el mateix text variant el vector d'orientació del caràcter ("character up vector") i augmentant el factor de separació entre caràcters, i augmentant l'alçada del caràcter.

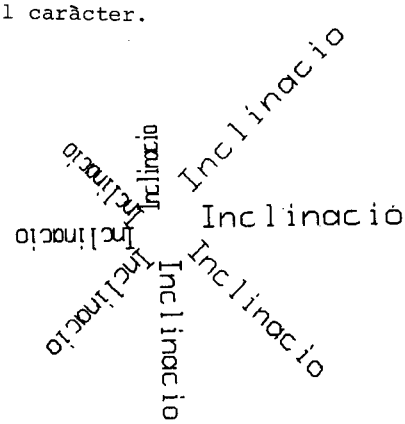


Fig. C: Primitiva "Text"

En la figura D es poden veure 4 "fillareas" en les quals els punts geomètrics coincideixen amb els de les figures A i B. S'han utilitzat 4 tipus d'omplenat d'àrea diferents.

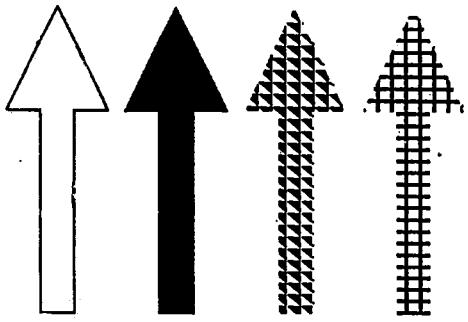


Fig. D: Primitiva "Fillarea"

En la figura E es pot veure una utilització de la primitiva "cellarray".

Fig. E: Primitiva "Cellarray"

En la figura F s'han dibuixat arcs, circles, sectors circulars i segments circulars utilitzant les primitives de dibuix generalitzat que hem implementat (GDP), es pot veure que aquestes primitives poden tenir associats els atributs de les primitives bàsiques de dibuix GKS, "polyline", "polymarker" i "fillarea".

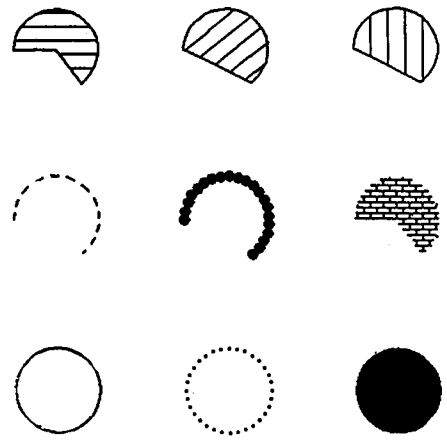


Fig. F: Primitives de dibuix generalitzat (GDP).

En la figura G es combina la utilització de la primitiva "text" i les primitives de dibuix generalitzat (sector circular).

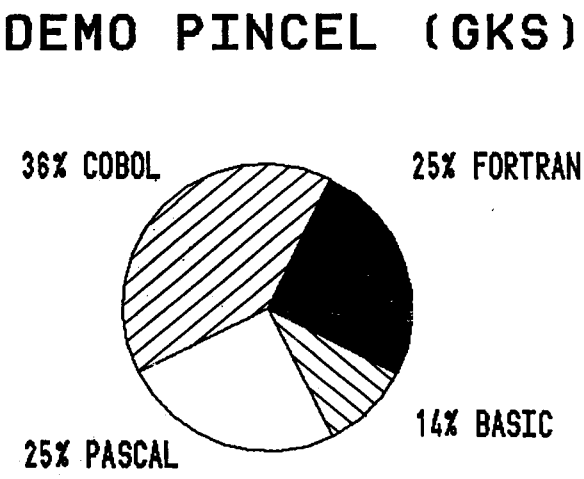


Fig. G: Primitives "Text" i GDP

11. EXEMPLES D'UTILITZACIO

BOTIGA

Dibuix d'una botiga de computadors mitjançant les primitives de dibuix TEXT, POLYLINE, FILLAREA i GDP (segment circular i sector circular).

A aquest exemple es pot veure l'aplicació de la rutina de dibuix FILLAREA en tres dels seus tipus d'omplenat: HOLLOW, SOLID i PATTERN.

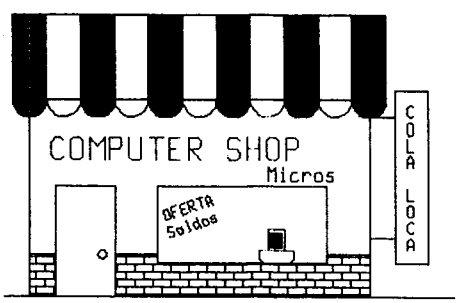
En la retolació del dibuix s'utilitza la rutina TEXT (GTX), modificant-se els seus atributs en cada cas. Fixeu-vos com, a més del tamany de caràcter, s'utilitza la variació

de la inclinació del text i la separació entre lletres, amb la modificació del sentit de dibuix s'aconsegueix dibuixar sense cap dificultat el rètol publicitari vertical al costat de la paret de l'edifici.

```
PROGRAM  botiga (INPUT, OUTPUT);
TYPE
%INCLUDE '$$:GKSTYPES, ESD/NOLIST'
VAR
id_wks, id_conexion,
wks_type                  : nom ;
i, j                      : INTEGER;
texto                     : VARYING [25] OF CHAR :
data                      : ARRAY [1..2] OF INTEGER ;
puertax, puertay          : ARRAY [1..4] OF REAL ;
basex, basey              : ARRAY [1..4] OF REAL ;
arcx, arcy                : ARRAY [1..3] OF REAL ;
terrax, terray            : ARRAY [1..2] OF REAL ;
pomx, pomy                : ARRAY [1..2] OF REAL ;
tejadox, tejadoy          : ARRAY [1..4] OF REAL ;
ventanax, ventanay        : ARRAY [1..4] OF REAL ;
casax, casay              : ARRAY [1..4] OF REAL ;
retolx, retoly            : ARRAY [1..5] OF REAL ;
uniorx, uniory            : ARRAY [1..4] OF REAL ;
microx, microy            : ARRAY [1..6] OF REAL ;
monitorx, monitory        : ARRAY [1..4] OF REAL ;
pantallax, pantallay      : ARRAY [1..4] OF REAL ;

VALUE
tejadox  := ( 80.0 , 120.0 , 120.0 ,  80.0);
tejadoy  := (280.0 , 280.0 , 380.0 , 380.0);
casax    := (100.0 , 100.0 , 500.0 , 500.0);
casay    := (280.0 ,  50.0 ,  50.0 , 280.0);
basex    := (100.0 , 100.0 , 500.0 , 500.0);
basey    := (100.0 ,  50.0 ,  50.0 , 100.0);
puertax  := (130.0 , 130.0 , 200.0 , 200.0);
puertay  := ( 50.0 , 180.0 , 180.0 ,  50.0);
ventanax := (250.0 , 250.0 , 450.0 , 450.0);
ventanay := ( 90.0 , 180.0 , 180.0 ,  90.0);
arcx     := (100.0 , 120.0 ,  80.0);
arcy     := (280.0 , 280.0 , 280.0);
terrax   := (  0.0 , 640.0);
terray   := ( 50.0 ,  50.0);
pomx     := (185.0 , 190.0);
pomy     := (100.0 , 100.0);
retolx   := (530.0 , 570.0 , 570.0 , 530.0 , 530.0);
retoly   := (290.0 , 290.0 ,  90.0 ,  90.0 , 290.0);
uniorx   := (530.0 , 500.0 , 500.0 , 530.0);
uniory   := (260.0 , 260.0 , 120.0 , 120.0);
monitorx := (380.0 , 380.0 , 400.0 , 400.0);
monitory := (105.0 , 130.0 , 130.0 , 105.0);
pantallax:= (383.0 , 383.0 , 397.0 , 397.0);
pantallay:= (108.0 , 127.0 , 127.0 , 108.0);
microx   := (375.0 , 370.0 , 370.0 , 410.0 , 410.0, 405.0);
microy   := ( 90.0 ,  95.0 , 105.0 , 105.0 ,  95.0,  90.0);
```

```
% INCLUDE '$$: Declarel.pas/list'

BEGIN
{---Apertura del GKS i de l'estació de treball ---}

id_wks    :='uno
READLN(id_conexion);
READLN(wks_type);

open_gks ;
open_workstation (id_wks, id_conexion,wks_type) ;
activate_workstation (id_wks) ;
clear_workstation (id_wks) ;
{--- Definició de les finestres de dibuix ---}
set_window    (1 , 70.0 , 600.0 , 40.0 , 429.2)
set_viewport (1 , 0.0 , 0.640 , 0.0 , 0.470);
select_normalization_trans (1) ;

{--- Dibuix dels elements que composen la tenda ---}

set_fillarea_representation (id_wks,1,pattern,3,3) ;
fillarea (4, basex[1] , basey[1] );
set_fillarea_representation (id_wks,1,hollow,3,3);
fillarea (4, basex[1] , basey[1] ) ;
set_fillarea_representation (id_wks,1,solid,3,0);
fillarea (4,puertax[1] , puertay[1] ) ;
polyline (4,puertax[1] , puertay[1] ) ;
polyline (4, casax[1] , casay[1] ) ;
fillarea (4, ventanax[1] , ventanay[1] ) ;
set_fillarea_representation (id_wks,1,hollow,3,3) ;
fillarea (4, ventanax[1] , ventanay[1] ) ;

FOR i:= 1 TO 5 DO
  BEGIN
  set fillarea_representation (id_wks,1,solid,3,3) ;
  fillarea (4 , tejadox 1   , tejadoy[1] ) ;
  gdp (3 , arcx[1] , arcy[1] , 23 , data) ;
  set fillarea_representation (id_wks,1,hollow,3,3) ;
  fillarea (4, tejadox[1] , tejadoy[1] ) ;
  FOR j: 1 TO 4 DO tejadox[j] := tejadox[j] + 40.0 ;
  FOR j: 1 TO 3 DO arcx [j] := arcx[j] + 40.0 ;
  fillarea (4 , tejadox[1] , tejadoy[1] ) ;
  gdp (3 , arcx[1] , arcy[1] , 24 , data) ;
  FOR j:= 1 TO 4 DO tejadox[j] := tejadox[j] + 40.0 ;
  FOR j:= 1 TO 3 DO arcs [j] := arcx[j] + 40.0 ;
  END ;
```

```
set_fillarea_representation (id_wks,1,solid,3,3) ;
fillarea (4 , tejedox 1  , tejedoy 1 ) ;
gdp (3 , arcx 1  , arcy 1  , 24 , data) ;
set_fillarea_representation (id_wks,1,hollow,3,3);
fillarea (4, tejadox 1  , tejadoy 1  ) ;

polyline (2 , terrax 1  , terray 1  ) ;
gdp (2 , pomx 1  , pomy 1  , 11 , data ) ;

polyline (6 , microx 1  , microy 1  ) ;
polyline (4 , monitorx 1  , monitory 1  ) ;
set_fillarea_representation (id_wks,1,solid,3,3) ;
fillarea (4, pantallax 1  , pantallay 1  ) ;
 --- Dibuix dels diferents r`etols ---
set_character_spacing (0.4) ;
set_character_height (0.030 1.73 530) ;
READLN (texto) ;
gtx (125.0 , 210.0 , texto) ;

set_character_height (0.018 1.73 530) ;
READLN (texto) ;
gtx(380.0 , 185.0 , texto) ;

set_character_height (0.0135 1.73 530) ;
set_character_up_vector (-1.0 , 3.0) ;
READLN (texto) ;
gtx (260.0 , 140.0 , texto) ;
READLN (texto) ;
gtx (265.0 , 120.0 , texto) ;

polyline (5 , retolx 1  , retoly 1  ) ;
polyline (4 , uniorx 1  , uniory 1  ) ;

set_character_path ( down ) ;
set_character_up_vector (0.0 , 1.0) ;
set_character_height (0.017 1.53 530) ;
READLN (texto) ;
gtx ( 545.0 , 260.0 , texto ) ;
 --- Tancament de l'estació de treball i del GKS ---
deactivate_workstation (id_wks) ;
close_workstation (id_wks) ;
close_gks
END
```



Sortida gràfica al primer exemple

En aquest exemple es mostren algunes de les possibilitats que ofereix el manegament de diferents finestres de visualització (WINDOWS i VIEWPORTS) i d'"escapçat" de les primitives de dibuix segons aquestes finestres (CLIPPING).

Es dibuixa tres cops un mateix triangle, variant a cada cas la transformació de normalització (conjunt de VIEWPORT i WINDOW) diferint únicament en la VIEWPORT escollida, amb l'objecte de dibuixar cada triangle a una zona distinta de la pantalla.

El primer rectangle es dibuixa amb l'indicador de CLIPPING en "clip". Degut a això, les puntes del triangle que resten fora del rectangle definit per la WINDOW són escapçats i per tant no es dibuixen.

El segon triangle es dibuixa amb l'indicador de CLIPPING en "noclip", amb la qual cosa tot el triangle es dibuixa, sense escapsar-li cap punta.

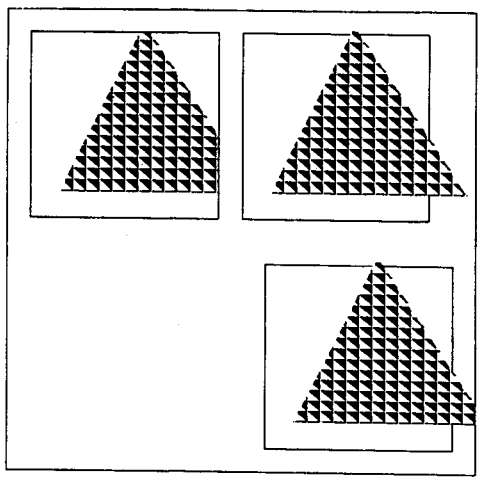
El tercer i darrer triangle es dibuixa en les mateixes condicions que l'anterior, ara bé, aquest cop la VIEWPORT seleccionada fa que part del triangle quedi fora del limits fixats per la pantalla, amb la qual cosa --- aquesta part no queda representada.

```
PROGRAM   TRIANGLE    (input,output) ;
TYPE
%INCLUDE '$$:GKSTYPES.ESD/nolist'

VAR
triax, triay                        : ARRAY [1..4] OF REAL;
bx, by, px, py                      : ARRAY [1..5] OF REAL;
estacion, conector, tipoestación    :                  nom;

VALUE
bx      :=(0.0, 1.0, 1.0, 0.0, 0.0):
by      :=(0.0, 0.0, 1.0, 1.0, 0.0);
triax   :=(-7.0, 2.0, 14.0, -7.0);
triay   :=(-7.0, 11.0, -7.0, -7.0);
px      :=(-10.0, -10.0, 10.0, 10.0, -10.0);
py      :=(-10.0, 10.0, 10.0, -10.0, -10.0);

% INCLUDE '$$:DECLARE1.PAS/NOLIST'
```

```
BEGIN
{--- Apertura del GKS i de l'estació de treball ---}
open_gks;
READLN (estación) ;
READLN (conector) ;
READLN (tipoestación) ;
open_workstation (estación, conector, tipoestación) ;
activate_workstation (estación) ;
clear_workstation    (estación) ;
{--- Definició de les finestres de visualització ---}
set_window (1, -10.0, 10.0, -10.0, 10.0) ;
set_window (2, -10.0, 10.0, -10.0, 10.0) ;
set_window (3, -10.0, 10.0, -10.0, 10.0) ;

set_viewport (1, 0.05, 0.45, 0.55, 0.75) ;
set_viewport (2, 0.5,  0.9 , 0.55, 0.95) ;
set_viewport (3, 0.55, 0.95, 0.05, 0.45) ;

set_fillarea_index (1) ;

fillarea (5, bx[1], by[1]) ;

{--- Dibuix del primer triangle ---}

select_normalization_trans (1) ;

fillarea (5, px[1], py[1]) ;

set_clipping_indicator (clip) ;

polymarker (3, triax[1], triay[1]) ;

set_fillarea_index (3) ;

fillarea (4, triax[1], triay[1] ;

{--- Dibuix del segon triangle ---}

select_normalization_trans (2) ;

set fillarea_INDEX (1) ;

fillarea (5, px[1], py[1]) ;

set_clipping_indicator (noclip) ;

polymarker (3, triax[1], triay[1]) ;

set_fillarea_index (3) ;

fillarea (4, triax[1], triay[1]) ;

{--- Dibuix del tercer triangle ---}

select_normalization_trans (3) ;

fillarea (5, px[1], py[1]) ;

set_clipping_indicator (noclip) ;

polymarker (3, triax[1], triay[1]) ;

set_fillarea_index (3) ;
```

```
fillarea (4, triax[1], triay[1]) ;
{--- Tancament de l'estació de treball i del GKS ---}
deactivate_workstation (estación) ;
close_workstation (estación) ;
close_gks ;

END
```

RESULTAT OBTINGUT.



Sortida gràfica al segon exemple.

12. CONCLUSIONS.

L'aportació més important del GKS en les aplicacions gràfiques consisteix en ser acceptada internacionalment com a estàndard (ISO i ANSI), la qual cosa facilita enormement la transportabilitat de les aplicacions.

Actualment s'està treballant en una extensió del GKS en 3D.

L'aparició de l'estandard software GKS insinua un nivell de funcionalitat ideal que cal que incorpori la nova perifèria gràfica. Seria molt interesant que s'establís un estàndard funcional per a l'anomenada perifèria basat en els actuals nivells del GKS.

El concepte de nivell d'implementació del GKS caldria que fos associat no a la implementació en sí sinó a cada estació de treball.

Per exemple el concepte de segment no és simulable de forma eficient mitjanç software, car els processos típics associats als segments com el de 'redibuixat' es converteixen en un reenviament de les comandes de dibuix la qual cosa és sempre ineficient.

13. BIBLIOGRAFIA.

/1/ "Information Processing Graphical Kernel System (GKS)" . Draft International Standard ISO/DIS. - Functional description. January 1982.

/2/ P.R. BONO, J.L. ENCARNAÇAO, F.R. HOPGOOD, P.J. ten HAGEN :GKS-The First Graphics Standard" - IEEE Computer Graphics & Applications. July 1982.

/3/ "Information Processing - Graphical Kernel System (GKS). Functional description". Draft International Standard ISO/ DIS. 1983.

/4/ J.L. ENCARNAÇAO, E.G. SCHLECHTENDAHL : "Computer Aided Desing. Fundamentals and System Architectures". Springer Verlag. Berlin 1983.

/5/ F. HOPGOOD, D. DUCE, J. GALOP, D.SUTCLIFFE : "Introduction to the Graphical Kernel System (GKS)". Academix Press. Londres 1983.

/6/ Q. ENDERLE, K. KANSY, Q. PFAF :"Computer Graphics Programming. GKS-The Graphics Standard". Springer-Verlag.Berlin 1984.

/7/ D.R. ROGERS, J.A. ADAMS : "Mathematical Elements for Computer Graphics". MCGraw-Hill. 1976.

/8/ TECHNICAL COMMITEE X3H3. ACM SIGGRAPH. "Computer Graphics GKS Issue" . February 1984.

/9/ J. ARESPACOCHAGA, E. TORRES, M.D. VICENTE, R. VILÀ : "Manual del Usuario del GKS" . U.P.C. 1984.